New Approaches to Practical Secure Two-Party Computation

Peter Sebastian Nordholt, 20022601

PhD Dissertation, Computer Science

February 2013

Advisor: Jesper Buus Nielsen

AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Abstract

We present two new approaches to maliciously secure two-party computation with practical efficiency:

- First, we present the first maliciously secure two-party computation protocol with practical efficiency based on the classic semi-honest protocol given by Goldreich et al. at STOC 1987. Before now all practical protocols with malicious security were based on Yao's garbled circuits.

  We report on an implementation of this protocol demonstrating its high efficiency. For larger circuits it evaluates 20000 Boolean gates per second. As an example, evaluating one oblivious AES encryption (around 34000 gates) takes 64 seconds, but when repeating the task 27 times it only takes less than 3 seconds per instance.

- Second, we revisit the LEGO protocol of Nielsen and Orlandi presented at TCC 2009. Their protocol demonstrated a more efficient technique to get malicious security in secure two-party computation protocols based on Yao's garbled circuit. Namely, doing the cut-n-choose test on the gate level instead of the circuit level. This idea speeds up the protocol by a factor the logarithm of the size of the circuit to be evaluated. The resulting protocol, however, was not considered practically efficient as it relies on public-key operations for every gate of the circuit.

  We demonstrate how to get rid of this dependency on public-key operations by replacing them with inexpensive Minicrypt type primitives. The resulting protocol maintains the LEGO protocols good asymptotic complexity, hopefully yielding a protocol of high practical efficiency.

- As a bi-product of these two new protocols for secure two-party computations we develop two new cryptographic tools of independent interest: for the first protocol we give a highly practical OT-extension protocol that, apart from a few OTs to bootstrap the construction, only needs 14 calls to hash function for each OT. For the second protocol we develop a new XOR-homomorphic commitment scheme based on OT.

Resumé

Vi præsenterer to nye tilgange til aktivt sikre topartsberegninger med praktisk effektivitet:

- Først præsenterer vi den første aktivt sikre topartsberegnings protokol med praktiske effektivitet baseret på den klassiske passivt sikre protokol givet af Goldreich et al. ved STOC 1987. Før nu var alle praktiske protokoller med aktiv sikkerhed baseret på Yao's forvanskede kredsløb.

  Vi rapporterer om en implementation af denne protokol der demonstrerer den høj effektivitet. For større kredsløb evaluerer denne 20.000 Boolske *gates* per sekund. Som et eksempel, evaluering af en AES kryptering (omkring 34.000 *gates*) tager 64 sekunder, og når vi gentage opgaven 27 gange tager det mindre end 3 sekunder per instans.

- For det andet, reviderer vi LEGO protokollen af Nielsen og Orlandi præsenteret til TCC 2009. Deres protokol viste en mere effektiv teknik til at få aktiv sikkerhed i topartsberegninger protokoller baseret på Yao's forvanskede kredsløb. Nemlig, at gøre skær-og-vælg testen på *gate* niveau i stedet for på kredsløbs niveau. Denne ide gør protokollen hurtigere med en faktor logaritmen af størrelsen af kredsløbet, der skal evalueres. Den resulterende protokol blev i imidlertid ikke anset for værende praktisk effektiv, da den bygger på offentlig nøgle operationer for hver *gate* af kredsløbet.

  Vi viser, hvordan man kan slippe af med denne afhængighed af offentlig nøgle operationer ved at erstatte dem med billige primitiver af Minicrypt typen. Den resulterende protokol fastholder LEGO protokollens gode asymptotiske kompleksitet, og giver forhåbentligt en protokol for høj praktiske effektivitet.

- Som et bi-produkt af disse to nye protokoller for sikre topartsberegninger, udvikler vi to nye kryptografiske værktøjer af uafhængig interesse: for den første protokol giver vi en meget praktisk OT-udvidelse protokol, bortset fra nogle få OTs til at starte konstruktionen, behøver vi kun 14 kald til en hash-funktion for hver OT. For den anden protokol vi udvikler en ny XOR-homomorf *commitment* protokol, der er baseret på OT.

Acknowledgements

First of all I would like to thank Jesper Buus Nielsen. For encouraging me to apply for a phd position in the first place, and for being a great advisor these past three years.

I would also like to thank the other people that have co-authored papers with me: Rikke Bendlin, Sebastian Faust, Tore Kasper Frederiksen, Carmit Hazay, Thomas Pelle Jakobsen, Claudio Orlandi, Angela Zottarel, and Sai Sheshank Burra. I have enjoyed working with all of them so very much.

Finally I would like to thank the people who has been part of the krypto-group at Aarhus University during my studies here. This is an absolutely amazing place to work, and I would like thank them all making it so.

Peter Sebastian Nordholt,
Aarhus, February 14, 2013.

Contents

Chapter 1

Introduction

In this dissertation we present two new protocols for secure two-party computation against malicious adversaries resulting from my phd studies. The focus has been on attaining protocols with good practical efficiency, and on finding alternative approaches to those commonly found in the known literature.

In the first part of the dissertation we give a protocol for secure two-party computation against a malicious adversary, based on the semi-honestly secure protocol of Goldreich et al. [GMW87]. Such a protocol was previously not thought to be practical as the underlying protocol relies heavily on oblivious transfer, and has a relatively high round complexity. However, by first giving a new and very practical protocol for maliciously secure extension of oblivious transfer and using message authentication codes to gain malicious security, we end up with a very practical protocol. To back up this claim we report on benchmarks of an implementation of the protocol showing that our new protocol is competitive with all known implementation of secure two-party computation. This part of the dissertation is joint work with Nielsen, Orlandi and Burra [NNOB12].

The second part of the dissertation revisits the LEGO construction introduced by Nielsen and Orlandi [NO09]. Their protocol gives a new and more efficient way of gaining security against a malicious adversary from Yao's [Yao86, LP09] semi-honestly secure protocol. They achieve this result by using the cut-n-choose technique in a more economical way. While the LEGO protocol demonstrated a very good asymptotic complexity it has been viewed as impractical, as it heavily relies on the use of expensive asymmetric cryptography. We show how to replace the use of asymmetric primitives with inexpensive symmetric ones, relying in part on the oblivious transfer extension result of the first part of the dissertation. At the time of writing we have not yet implemented the resulting protocol, but we are hopeful that it can lead to very practical implementations of secure two-party computation. This part of the dissertation is joint work with Frederiksen, Jakobsen, Nielsen and Orlandi [FJN+13].

In this chapter will start by giving a short introduction to secure two party computation, and then give more in-depth introductions to the two protocols presented in the dissertation.

1.1 Secure Two-Party Computation

Consider Alice and Bob each holding an inputs x and y respectively. Suppose Alice and Bob wants to compute some function f of their inputs. The task is easily solved: Alice sends x to Bob, Bob computes $z = f(x, y)$ and sends z to Alice. However, suppose that the input x is private to Alice, or there is mistrust between Alice and Bob. In that case the above solution is hardly very satisfying.

Consider for example a situation where Alice and Bob represent companies who want to know if they share customers. Each have a list of their private customers, and they want to compute the intersection of these two lists. In this case Alice may prefer to only reveal those customers she has in common with Bob and not *all* of her customers. Furthermore, as Alice could be competing with Bob, she may not fully trust that Bob will give her the correct result.

The aim of *secure two-party computation* is to solve this type of problem. I.e. to allow Alice and Bob to jointly compute any function f in such a way that 1) the parties learn nothing about the private input of the opposing party, except what can be directly inferred from $f(x, y)$, and 2) if an output z is given to any of the parties, then that is correct output, i.e., the output is $z = f(x, y)$ and not some other value $z' \neq f(x, y)$.

In this dissertation we only consider a so called *malicious* or *active* adversary. This means that a dishonest party can deviate from the protocol in any way she wishes. Note that this means that we cannot give any guarantee that the protocols will terminate they will not terminate prematurely. To see this consider an adversary that simply chooses to not participate in the protocol.

An different type of adversary that is also often considered is a so called *semi-honest* or *passive* adversary. Such an adversary follows the protocol faithfully, but uses the information she obtains from the protocol to try to compute some information she is not supposed to know. Such an adversary seems much less natural than a malicious one. However, as we shall see, protocols secure against semi-honest adversaries are often a valuable stepping stone towards protocols with malicious security.

In this dissertation we will also restrict our selves to functions f that are described as a Boolean circuit C. An alternative is to consider functions described as arithmetic circuits. While recent protocols and implementations has shown impressive results for this setting [BDOZ11, DPSZ12, DKL+12], we will not discuss it further as it lies outside the scope this dissertation.

1.2 Practical Secure Two-Party Computation

Secure two-party computation (2PC) was first introduced in 1982 by Yao [Yao82], who also gave the first protocol, albeit only with semi-honest security. The protocol essentially consists of Alice (A) sending an encrypted version of the circuit C to Bob (B) along with an encrypted version of her input. Bob then evaluates the encrypted circuit, and obtains an encrypted output, which is then decoded by A. In 1987 Goldreich et al. [GMW87] gave a conceptually very different

protocol. In this protocol A and B first secret share their inputs among each other. For each gate of C A and B then run a small protocol to obtain a secret sharing of the output of the given gate. After doing this for all gates A and B can securely reconstruct the output. This base protocol is also only semi-honestly secure. However, by essentially requiring both parties to give zero-knowledge proofs that they performed each step correctly, Goldreich *et al.* achieved malicious security.

While these early results of Yao and Goldreich *et al.* are mainly viewed as feasibility results, recently great attention has been given to developing protocols with practical efficiency (see e.g., [ST04, MNPS04, LP07, BDNP08, IPS08, LPS08, KS08, IKOS08, NO09, PSSW09, HKS+10, LP11, LOP11, SS11, HEK+11, KSS12, DKL+12, NNOB12, HKE12, BHR12] and references therein). Interestingly, most (if not all) protocols for malicious security still uses a pattern similar to that of Goldreich *et al.*: namely they get malicious security by starting from a semi-honestly secure protocol, and then add some mechanism to force a malicious adversary to follow the protocol. Since the underlying semi-honest protocols are usually quite efficient, much of the effort in producing maliciously secure protocols goes into reducing the overhead associated with enforcing malicious security. Unlike Goldreich *et al.*, most practical protocols no longer use zero-knowledge proofs to enforce malicious security. Instead most protocols now use the more efficient *cut-n-choose* technique, first used for 2PC in 2007 by Lindell and Pinkas [LP07]. We describe this technique in more detail below.

Asymmetric primitives such as public-key encryption, are typically orders of magnitude more computationally expensive than symmetric primitives, such as hash functions or one-way permutations. Therefore, a lot of focus of practical 2PC protocols is on reducing the use of – or all together replacing – asymmetric primitives with cheap symmetric primitives. Consequently the underlying semi-honest protocol most commonly used is Yao's protocol, described above, as it almost exclusively uses symmetric primitives.

The protocols presented in this dissertation share this focus on both reducing the overhead of achieving malicious security and reducing the number of expensive asymmetric primitives used in the protocol. All though, as we discuss in the following section we slightly change our view on what primitive we consider to be expensive.

Very recently a few works has come out that focuses strongly on how to parallellize 2PC protocols [KSS12, FN13]. These works use massively parallelized hardware, such as GPU's and computer clusters, to gain large speed ups of their implementations. Work in this direction seems very promising, and has so far resulted in the fastest known implementations of 2PC. The protocols presented here has not explicitly been developed with this focus in mind. However, they do have do have characteristics that would allow them to be easily parallelized, so it is likely that they to could be sped up with this approach.

1.3 Our Techniques

Here we describe the main techniques our protocols have in common.

1.3.1 OT-Extension

We abandon the view of oblivious transfer as being an expensive primitive, and view it instead as being as cheap as symmetric primitives such as hash-functions. Consequently we allow our protocols to heavily rely on OT. We justify this view using the technique of OT-extension:

All known implementations of OT requires expensive public-key primitives, and there is evidence to suggest that we can not base OT solely on cheap symmetric primitives. E.g. [IR89] shows that a black box reduction of OT to one-way functions (or one-time permutations) would imply a proof that P $\neq$ NP, meaning that it will likely be very hard to find such a reduction (as it would be an amazing breakthrough for all of computer science). Therefore, OT has traditionally been viewed as an expensive primitive. However, in [Bea96] Beaver showed that *given* a small number of initial OTs we can implement a much larger number of OTs using only symmetric primitives. We call this process OT-extension. I.e. while OT seems to require public-key cryptography, it does not require the use of public-key operations for *every* OT needed in a protocol.

While Beaver's initial result was mainly of theoretical interest and not practically efficient, subsequent works has greatly reduced the complexity of the OT-extension technique [IKNP03, Nie07, HIKN08]. Indeed one of the contributions of this dissertation (presented in Chapter 3) is an OT-extension protocol that, while matching the best known asymptotic complexity of Harnik et al. [HIKN08], is perhaps the most efficient maliciously secure OT-extension protocol known in terms of practical efficiency. Beyond the few OTs needed to bootstrap, this protocol requires only a small constant of calls to a hash function (modeled as a random oracle) per additional OT. This means that for any protocol using a large amount of OTs we can easily amortize away the cost of the public-key primitives used in the initial OTs, thus justifying our view of OT as essentially a symmetric-primitive in terms of complexity.

1.3.2 LEGO-Style Cut-n-Choose

To get active security techniques similar to the LEGO style cut-n-choose technique introduced by Nielsen and Orlandi [NO09]. This technique improves on the standard cut-n-choose technique, as seen in many Yao-based protocols, in the following way.

On an abstract level the cut-n-choose technique works in two phases: First we semi-honestly produce a set of gadgets G. I.e., the gadgets in G are secure if both parties are semi-honest, but could be insecure if one party is malicious. Second we randomly partition the gadgets into two sets, the *usable set* U, and the *test set* T. We then perform a test on each gadget in T that will detect if that gadget was generated dishonestly. If any gadget in T does not pass the test we know that one party is corrupted and we can safely abort the protocol. On the other hand, if all gadgets in T passes the test, then with probability $1 - 2^{-O(k)}$ at most k gadgets in U are dishonestly generated. If we let k be the statistical security parameter we can assume that all but k gadgets in U are

secure. We then use some method to combine the gadgets in U into a smaller set of gadgets S, which will be maliciously secure given the above assumption.

One way to measure the efficiency of using the cut-n-choose method is the *replication factor* $r = |G|/|S|$. This measures the number of semi-honest gadgets we need to produce in the first phase, for each maliciously secure gadget resulting from the second phase. In other words, the replication factor gives us a measure of the overhead involved in getting malicious security compared to semi-honest security.

The standard way of using the cut-n-choose technique in 2PC is to let the gadgets represent a secure version of the circuit to be evaluated C. I.e., U will be a set of semi-honestly secure circuit-gadgets where at most k are dishonestly generated. To produce *one* maliciously secure circuit-gadget from the gadgets in U, we simply evaluate all circuit-gadgets in U and take the majority result. I.e., we have $|S| = 1$ and $r = |G|$ and the replication parameter must be such that $|U| > 2k$ and this means we must have $r = O(k)$. As the work to produce one circuit-gadget is proportional to the size of the circuit, the overall complexity becomes $O(r|C|) = O(k|C|)$.

On the other hand in the LEGO construction we let the gadgets represent a secure version of gates. I.e. U becomes a set of semi-honestly secure gate-gadgets where at most k are dishonestly generated. We then combine the gadgets in U into $|S| = |C|$ maliciously secure gate-gadgets. These maliciously secure gate-gadgets are then used to evaluate a secure version of C. Since we need at least one semi-honest gate-gadget to produce one maliciously secure gate-gadget we have that $|U| \geq |C|$. However, the cut-n-choose test gives us that at most k of the $|U|$ gadgets are dishonest, where k is a statistical security parameter independent of $|C|$. Therefore, as $|C|$ grows the fraction of dishonest gate-gadgets in U decreases. Since the number of semi-honest gate-gadgets needed to implement one maliciously secure gate-gadget depend on this fraction, this means that as $|C|$ grows the replication factor decreases. Skipping many details this means we get a replication factor of $r = O(k/\log(|C|))$. Since the work to produce a gate-gadget is constant, the overall complexity using this technique becomes $O(r|C|) = O(k|C|/\log(|C|))$.

We note that the approach we use to get malicious security in the protocol described in the first part of the dissertation, can not strictly speaking be seen as a cut-n-choose variant. However, the approach is strongly inspired by the LEGO-style cut-n-choose technique, and has similar characteristics in terms of the replication factor and asymptotic complexity.

Concretely this means that we for a circuit of size $2^{20} \approx 1{,}000{,}000$, can get statistical security 2^{-40} with replication factor only 4 for our first protocol and 24 for our second protocol. In contrast in, e.g., shelat et al. [SS11], using the standard cut-n-choose technique, they must use a replication factor of 125 to get the same security level regardless of the circuit size.

1.4 Our Protocols

Here we give a more in depth introduction to the two 2PC protocols presented in this dissertation.

1.4.1 Part 1: TinyOT

The protocol presented in the first part of the dissertation we call the *TinyOT* protocol. The name originates from Jesper Buus Nielsen who, one day while examining the output of his implementation of the OT-extension protocol involved, exclaimed with joy "Look at all these tiny OT's!".

Our starting point is the efficient semi-honestly secure OT-extension protocol of Ishai *et al.* [IKNP03] and the semi-honestly secure 2PC protocol of Goldreich *et al.* [GMW87]. The protocol of Ishai *et al.* allows to turn ψ seed OTs based on public-key crypto into any polynomial $\ell = \text{poly}(\psi)$ number of OTs, using only $O(\ell)$ invocations of a cryptographic hash function. As mentioned above this means that, for big enough ℓ, the cost of the ψ seed OTs can be amortized away and OT extension essentially turns OT into a symmetric primitive in terms of its computational complexity. Meanwhile, the basic 2PC protocol of Goldreich *et al.* is efficient in terms of consumption of OTs and communication. Thus, the idea is to combine these two protocols to get highly practical 2PC. In order to gain malicious security and preserve the high practical efficiency of these protocols we develop substantially different techniques, differentiating from other works that were only interested in *asymptotic* efficiency [HIKN08, Nie07, IPS08]. Our contributions are the following:

1. We introduce a new technical idea to the area of extending OTs efficiently, which allows to dramatically improve the practical efficiency of active-secure OT extension. Our protocol has the same asymptotic complexity as the previously best known protocol in [HIKN08], but it is only a small factor slower than the passive-secure protocol in [IKNP03].

2. We introduce new technical ideas which allow to relate the outputs and inputs of OTs in a larger construction, via the use of information theoretic tags. This can be seen as a new flavor of committed OT that only requires symmetric cryptography. In combination with our first contribution, our protocol shows how to efficiently extend committed OT. Our protocols assume the existence of OT and are secure in the random oracle model.

3. We give the first implementation of practical 2PC not based on Yao's garbled circuit technique.

4. We implement all the above mentioned protocols: The implementation of our OT-extension protocol extends a few hundred maliciously secure OTs into tens of millions of OTs at a rate of around 500,000 maliciously secure OTs per second, demonstrating that implementations needing a large number of OTs can be practical. We use this to implement our 2PC protocol which also gives very competitive benchmarks compared to Yao-based implementations.

Technical Overview

To give our OT-extension protocol we first notice that it suffices to extend the seed OTs to a variant of OT we call Δ-OT. In Δ-OT all messages are correlated by a constant XOR Δ. I.e. the messages of the sender are always of the form M_0 and M_1 so that $M_0 \oplus M_1 = \Delta$. It then turns out, that when using the OT extender in [IKNP03] and starting from seed OTs where the XORs of message pairs are constant, one also produces OTs where the XORs of message pairs are constant, and for this use the protocol happens to be *maliciously* secure. I.e. if we can make sure that the XOR of message pairs in the seed OT's are correlated in this way, we can efficiently extend a few OTs to an essentially unbounded number of Δ-OTs. Then using a cut-and-choose like technique we ensure that most of the XORs of message pairs offered in the seed OTs are constant, and with a new and inexpensive trick we offer privacy and correctness even if few of these XORs have different values.

For our 2PC protocol we start from the protocol of Goldreich et al. Say A holds secret shares x_A and y_A and B holds secret shares x_B and y_B of some bits x and y s.t. $x_A \oplus x_B = x$ and $y_A \oplus y_B = y$. To compute a secret sharing of the AND $z = ab$, A and B need to compute a random sharing z_A, z_B of $z = xy = x_A y_A \oplus x_A y_B \oplus x_B y_A \oplus x_B y_B$. The parties can compute the AND of their local shares ($x_A y_A$ and $x_B y_B$), while they can use OT to compute the cross products ($x_A y_B$ and $x_B y_A$). On the other hand A and B can compute a secret sharing of the XOR $z = x \oplus y$, by simply letting A compute $z_A = x_A \oplus y_A$ and B compute $z_B = x_B \oplus y_B$ (note that this does not require any interaction between A and B). Now the parties can iterate for each gate of the circuit, until they have evaluated the entire circuit. Finally, they will reconstruct the output values by revealing their shares.

This protocol is secure against a semi-honest adversary: assuming the OT protocol to be secure, A and B learn nothing about the intermediate values of the computation. It is easy to see that if a large circuit is evaluated, then the protocol is not secure against a malicious adversary: any of the two parties could replace values on any of the internal wires, leading to a possibly incorrect output and/or leakage of information.

To cope with this, we put MACs on all bits. The starting point of our protocol is *oblivious authentication* of bits. One party, the *key holder*, holds a uniformly random *global key* $\Delta \in \{0,1\}^\psi$. The other party, the *MAC holder*, holds some secret bits (x, y, say). For each such bit the key holder holds a corresponding uniformly random *local key* ($K_x, K_y \in \{0,1\}^\psi$) and the MAC holder holds the corresponding *MAC* ($M_x = K_x \oplus x\Delta$, $M_y = K_y \oplus y\Delta$). The key holder does not know the bits and the MAC holder does not know the keys. Note that $M_x \oplus M_y = (K_x \oplus K_y) \oplus (x \oplus y)\Delta$. So, the MAC holder can locally compute a MAC on $x \oplus y$ under the key $K_x \oplus K_y$ which is non-interactively computable by the key holder. This homomorphic property comes from fixing Δ and we exploit it throughout our constructions. From a bottom-up look, our protocol is constructed as follows:

Bit Authentication: We first notice that oblivious authentication of bits (aBit), as described above is essentially the same as the Δ-OT we implemented as

part of our OT-extension protocol. As described above we can construct a virtually unbounded number of such Δ-OTs by extending a few seed OTs.

Authenticated local AND: From aBits we then construct *authenticated local AND*s (aAND), where the MAC holder locally holds random authenticated bits a, b, c with $c = ab$. To create authenticated local ANDs, we let one party compute $c = ab$ for random a and b and get authentications on a, b, c (when creating aANDs, we assume the aBits are already available). The challenge is to ensure that $c = ab$. We construct an efficient proof for this fact. This proof might, however, leak the bit a with small but noticeable probability. We correct this using a combiner.

Authenticated OT: From aBits we also construct *authenticated OT*s (aOT), which are normal $\binom{2}{1}$-OTs of bits, but where all input bits and output bits are obliviously authenticated. This is done by letting the two parties generate authenticated bits representing the sender messages x_0 and x_1 and the receiver choice bit c. To produce the receiver's output, first a random authenticated bit is sampled. Then this bit is "corrected" in order to be consistent with the run of an OT protocol with input messages x_0 and x_1 and choice bit c. This correction might, however, leak the bit c with small but noticeable probability. We correct this using an OT combiner.

2PC: Given two aANDs and two aOTs one can evaluate in a very efficient way any Boolean gate: only 4 bits per gate are communicated, as the MACs can be checked in an amortized manner.

That efficient 2PC is possible given enough aBits, aANDs and aOTs is no surprise. In some sense, it is the standard way to base passive-secure 2PC on passive-secure OT enhanced with a particular flavor of committed OT (as in [CvdGT95, GMY04]). What is new is that we managed to find a particular committed OT-like primitive which allows both a very efficient generation and a very efficient use: while previous results based on committed OT require hundreds of *exponentiations* per gate, our cost per gate is in the order of hundreds of *hash functions*. To the best of our knowledge, we present the first practical approach to extending a few seed OTs into a large number of committed OT-like primitives. Of more specific technical contributions, the main is that we manage to do all the proofs efficiently, thanks also to the preprocessing nature of our protocol: Creating aBits (Δ-OTs), we get active security paying only a constant overhead over the passive-secure protocol in [IKNP03]. In the generation of aANDs and aOTs, we use efficient proofs that are similar to the LEGO-style cut-n-choose technique: when we preprocess for ℓ gates and combine B leaky objects to get an unleaky object, the probability of protocol failing is $(2\ell)^{-B} = 2^{-\log_2(\ell)(B-1)}$. Thus for statistical security parameter k we get negligible probability of failure when $B = O(k/\log(\ell))$. As an example, if we preprocess for 2^{20} gates with an overhead of $B = 6$, then we get leakage probability 2^{-100}.

	Protocols	Security	Model	Rounds	Time
(a)	[SS11]	Active	SM	$O(1)$	192s
(b)	[HEK+11]	Passive	ROM	$O(1)$	0.2s
(c)	[IPS08, LOP11]	Active	SM	$O(d)$	79s
(d)	TinyOT (single)	Active	ROM	$O(d)$	64s
(e)	TinyOT (27, amortized)	Active	ROM	$O(d)$	2.5s
(f)	[FN13]	Active	ROM	$O(1)$	2s
(g)	[KSS12] (One core)	Active	SM	$O(1)$	115s
(h)	[KSS12] (265 cores)	Active	SM	$O(1)$	1.4s

Table 1.1: Brief comparison with other implementations.

Comparison with Related Work

A brief comparison of the time needed for oblivious AES evaluation for the best known implementations are shown in Table 1.1.[1] The column *Round* indicates the round complexity of the protocols, d being the depth of the circuit while the column *Model* indicates whether the protocol was proved secure in the standard model (SM) or the random oracle model (ROM).

Row (a) shows the previous fastest protocol with malicious security before ours. Row (b) shows the time for a state of the art protocol with semi-honest security to demonstrate the low overhead in getting malicious security. Row (c) is an estimate made by [LOP11] on the running time of their optimized version of the OT-based protocol in [IPS08].

The performance of our protocol is shown in row (d) and (e). The reason for the dramatic drop between row (d) and (e) is that in (d), when we only encrypt one block, our implementation preprocesses for many more gates than is needed, for ease of implementation. In (e) we encrypt 27 blocks, which is the minimum value which eats to up all the preprocessed values. In this case the time of 2.5 seconds is *per* AES block.

In row (f), (g) and (h) we show the performance of some more recent implementations based on Yao's protocol.

We stress that the comparison is inherently unfair, as the different experiments were all run on different hardware, network, with different security parameters and so on. As an example: the results in row (f), (g) and (h) were run on massively parallelized hardware. The significance of running on such hardware can be seen in the difference between row (g) and (h) running on 1 and 256 cores respectively.

In spite of the experiments not being directly comparable, we conclude from the comparison that our protocol at least runs in *reasonable* time compared to the Yao based alternatives.

[1] Oblivious AES has become one of the most common circuits to use for benchmarking generic MPC protocols, due to its reasonable size (about 30000 gates) and its relevance as a building block for constructing specific purpose protocols, like private set intersection [FIPR05].

1.4.2 Part 2: MiniLEGO

The protocol in the second part of the dissertation we call *MiniLEGO*, as it is an implementation of the LEGO protocol of Nielsen and Orlandi [NO09] relying essentially only on primitives from Minicrypt.

As described above Nielsen and Orlandi in the LEGO protocol gave a twist on the standard approach for maliciously secure 2PC based on Yao's protocol: their approach consists of performing a cut-n-choose test at the gate level (instead of at the circuit level), and allows to save a factor $\log |C|$ in the computation and communication complexity w.r.t., standard cut-n-choose test at the circuit level. That is, A prepares many encrypted gates, B checks some of them and, if no inconsistency is detected, the unopened gates are "soldered" together in a redundant version of the original circuit that computes the right result even in the presence of a few faulty gates.

The LEGO approach has been praised for its novelty [Gol09], but did not have a practical impact for the efficiency of Yao-based protocols. There are several reasons for this:

1. *LEGO uses public-key primitives for each gate in the circuit:* Each gate has in fact associated three commitments to its input/output keys. Those commitments are used for the "soldering" and need to be homomorphic. For this purpose LEGO uses Pedersen commitments (based on the hardness of computing discrete logarithms in some group – e.g., the group of points of an elliptic curve). This is a drawback for the efficiency of the protocol (group operations, even in an elliptic curve, are orders of magnitude slower than symmetric crypto primitives such as hash functions or private-key encryption). Moreover, this is also a drawback as LEGO can only be implemented using a few computational assumptions – it is unclear how to implement LEGO under a variety of computational assumptions (including assumptions that are believed to withstand quantum attacks).

2. *LEGO is not compatible with known optimization for Yao's protocol:* Keys in LEGO are elements of $\mathbb{Z}_p$ for some prime p, while using binary strings $\{0,1\}^k$ is more natural and standard. Therefore, it is not possible to use the "free-XOR" trick with LEGO, nor many of the others optimizations that are tailored for bit-string keys.

3. *LEGO has too many bricks:* there are many different kind of objects in LEGO (key-filters, not-two gates, etc.) that make the use of LEGO complex to understand and implement.

Here we present a generalization and a simplification of the LEGO approach. The main technical difference is to replace the Pedersen commitments with XOR-homomorphic commitments based on OT. Doing so allows us to:

1. Maintain LEGO's good complexity and achieve security 2^{-k} when the replication factor is only $O(k/\log(|C|))$ against a replication factor of $O(k)$ for the standard cut-n-choose methods such as the one in [LP07].

2. Implement a variant on LEGO whose security only relies on generic, symmetric crypto primitives (except for the few seed OTs needed to bootstrap the OT extension).

3. Achieve a variant of LEGO that uses "standard" garbled gates (ANDs and free XORs), compatible with garbled gates optimization.

In conclusion, we propose the first real alternative to standard cut-n-choose for practical purposes Yao-based 2PC. Whether our proposed protocol will be more efficient in practice than protocols with standard cut-n-choose [LP07, PSSW09,SS11,KSS12] will only be decided by performing a serious comparison of similar implementations running on the same hardware-network configuration of our and other approaches. This is an interesting direction for future work.

Technical Overview

The main idea of the protocol we present is the same as in [NO09]: A prepares many garbled gates together with commitments to the input and output garbled keys. If A prepares a gate dishonestly we view it as a faulty gate, i.e. one that does not give the correct output on some inputs. B asks A to open a random subset of the gates and checks them for correctness. If the check goes through, B randomly permutes the unopened gates to form a redundant circuit that computes the function even in the presence of few faulty gates.

As the gates have been generated independently, the output keys of the gates in one layer of the circuit cannot be directly fed as input to the next layer. Therefore, A reveals the difference between the output keys in the first layer with the corresponding input keys in the second layer (using the XOR-homomorphic properties of the commitment scheme). This allows B to "align" the input keys of the gates in one layer with the output keys of the gates in the previous layer. The main intuition for the security of LEGO cut-n-choose is as follows: If A had sent B t faulty gates, B would detect this with probability $1 - 2^{-t}$. Therefore, if B accepts the test, with very high probability there are only a few faulty gates among the unopened ones. As all gates are permuted at random and placed in random positions in the circuit, only very little redundancy is needed to correct for all faulty gates.

The original LEGO construction in [NO09] used a set of specially tailored garbled gates and a complex design for the redundant circuit used to deal with faulty gates. This meant that it was not possible to apply standard garbling techniques and optimizations. In contrast, the construction here can be instantiated with essentially any free-XOR compatible garbled gate scheme and is compatible with various state of the art optimizations (such as free-XOR, row-reduction, point-and-permute). Additionally we can design the fault tolerant circuit in a very simple way: we implement each gate of the original circuit by replicating it ρ times, and take the output of the gate to be any output agreed upon by more than $\rho/2$ of the replicated gates. Thus we can tolerate up to $\rho/2$ faulty gates used to implement each gate of the original circuit.

The main technical contribution of this protocol is a novel construction of XOR homomorphic commitment based on OT. The main intuition is as follows:

let E be a linear error correcting code with the property that seeing a few bits of the codeword does not leak any information about the encoded message.

Then to commit to a set of messages $\{m_i\}_i$ A encodes her messages $c_i = E(m_i) \in \{0,1\}^n$ and inputs them into a $\binom{n}{t}$-$\mathcal{F}_{\text{OT}}$, so that the first bit of all c_i's are the first input to the OT, and so on. B chooses t positions in the OT and learns t bits of all codewords (note that B gets to see the same subset of bits for all codewords). We call this B's watch list, as it is reminiscent of the watch lists construction in [IKOS08, IPS08].

The commitment is intuitively hiding because, by the property of E, seeing only t position of c does not leak any information about m. At the same time as the code is linear $E(m_1) \oplus E(m_2) = E(m_1 \oplus m_2)$ and therefore it is possible to open the XOR of two committed messages without revealing any extra information about the original messages.

To argue that the commitment is binding we observe that, if A inputs actual codewords, she cannot later change her mind: if A wants to open the commitment c_i to a different message m_i', she needs to find an encoding of m_i' that agrees with c_i on the t positions seen by B during the OT. As the error correction property implies that the code has a high minimal distance, any encoding of m_i' will disagree with c_i in many positions and therefore B has a high probability of detecting cheating.

However, if A's input to the OT are not codewords, this argument does not hold. To make sure that A's is behaving honestly we will use a cut-n-choose style test: A commits to 2ℓ random messages and B checks half of them for consistency. In the analysis, we show that this implies that with high probability most of the unopened commitments are very close to actual codewords – this would be enough if commitments were to be opened one at the time since, given the error correcting property of the code, something close to a codeword is essentially a codeword for all practical purposes. But, as we want the commitments to be XOR-homomorphic (for an a unbounded number of times), even if every codeword only contains few "errors", when combining them together the number of errors could grow arbitrarily. In the analysis we carefully show that, due to the way our commitments are constructed, this does not happen.

The few "bad" commitments left will not influence the security of the overall protocol, as they will simply be absorbed by the analysis of the 2PC protocol which, as discussed before, due to the LEGO cut-n-choose and the use of a redundant circuit ensures privacy and correctness even in the presence of a few faulty gates.

1.5 Overview The Dissertation

In the following chapter we give a very short set of preliminaries and notation commonly used through out the dissertation. In part one we present the TinyOT protocol and in part two the MiniLEGO protocol.

For each of the two main protocols presented, we will be developing a new cryptographic tool: for TinyOT an OT-extension protocol and for MiniLEGO

a new type of XOR-homomorphic commitments. To ease the presentation of the 2PC protocols, and to give these tools the attention they deserve, we will factor their description out into separate chapters.

It has been our goal to make each chapter as self contained as possible, and it should be possible to understand them separately given the preliminaries in Chapter 2.

Chapter 2

Preliminaries

2.1 Notation

Sets and Strings

We let $[n] = \{1, \ldots, n\}$. For a finite set S we will use $|S|$ to denote the size of S. For a bit-string $S \in \{0,1\}^*$ let $|S|$ be the length of S, and we define $0S \stackrel{\text{def}}{=} 0^{|S|}$ and $1S \stackrel{\text{def}}{=} S$. For a subset $R \subseteq S$ we let $\bar{R}$ be the set $S \setminus R$ (when it is clear from context what set R is a subset of). We will use $(e_i)_{i \in I}$ to denote a *sequence* of elements e_i indexed by the set I, and we will use $\{e_i\}_{i \in I}$ to denote a *set* of elements e_i indexed by the set I.

Probabilities

For a finite set S we use $s \in_{\text{R}} S$ to denote that s is chosen uniformly at random in S. For a finite distribution D we use $x \leftarrow D$ to denote that x is sampled according to D.

We say that a function f is *negligible* in n and we write $f = \text{negl}(n)$ if for any polynomial p and large enough n $f(n) < 1/p(n)$ (this goes for any function, but usually a negligible function is used to describe vanishingly small probabilities). If g is a function describing a probability and $g(n) \geq 1 - f(n)$ where f is negligible in n we say that g is *overwhelming* in n. If for some polynomial p we have $g(n) \geq 1/p(n)$ for large enough n we say that g is noticeable in n. If we just say f is negligible, overwhelming or noticeable it is taken to mean in the security parameter.

Security Parameters

We use ψ to denote the computational security parameter and σ to denote the statistical security parameter. We require that a poly-time adversary break our protocols with probability at most $\text{poly}(\psi)2^{-\psi}$, and we allow our protocols to break with probability at most $2^{-\sigma}$ independent of the computational power of the adversary.

2.2 The UC Framework and Functionalities

We prove our results statically secure in the UC framework [Can01] against a malicious adversary. We assume the reader to be familiar with the UC framework. To simplify the statements of our results we use the following terminology:

Definition 2.1. We say that a functionality $\mathcal{F}_{\mathrm{A}}$ is *reducible* to a functionality $\mathcal{F}_{\mathrm{B}}$ if there exist an actively secure implementation π of $\mathcal{F}_{\mathrm{A}}$ which uses only one call to $\mathcal{F}_{\mathrm{B}}$. We say that $\mathcal{F}_{\mathrm{A}}$ is *locally* reducible to $\mathcal{F}_{\mathrm{B}}$ if the parties of π do not communicate (except through the one call to $\mathcal{F}_{\mathrm{B}}$). We say that $\mathcal{F}_{\mathrm{A}}$ is *linear* reducible to $\mathcal{F}_{\mathrm{B}}$ if the computing time of all parties of π is linear in their inputs and outputs. We use *equivalent* to denote reducibility in both directions.

It is easy to see that if $\mathcal{F}_{\mathrm{A}}$ is (linear, locally) reducible to $\mathcal{F}_{\mathrm{B}}$ and $\mathcal{F}_{\mathrm{B}}$ is (linear, locally) reducible to $\mathcal{F}_{\mathrm{C}}$, then $\mathcal{F}_{\mathrm{A}}$ is (linear, locally) reducible to $\mathcal{F}_{\mathrm{C}}$.

Hash Functions

We use a hash function $H : \{0,1\}^* \to \{0,1\}^\psi$, which we model as a random oracle (RO). We sometimes use H to mask a message, as in $H(x) \oplus M$. If $|M| \neq \psi$, this denotes $\mathrm{prg}(H(x)) \oplus M$, where prg is a pseudo-random generator $\mathrm{prg} : \{0,1\}^\psi \to \{0,1\}^{|M|}$. We also use a collision-resistant hash function $G : \{0,1\}^{2\psi} \to \{0,1\}^\psi$.

We typically use H for two task: 1) As a prg to extend a string $s \in_{\mathrm{R}} \{0,1\}^\psi$ to some longer string $s' \in \{0,1\}^\ell$ for some $\ell = \mathrm{poly}(\psi)$. 2) To hash a long string $s \in \{0,1\}^\ell$ to a short string for some $s' \in \{0,1\}^\psi$. When we attempt to sketch the complexity of these tasks we count both tasks as ℓ/ψ calls to the hash function H.

As other 2PC protocols whose focus is efficiency [KS08, HEK+11], we are content with a proof in the random oracle model. What is the exact assumption on the hash function that we need for our protocol to be secure, as well as whether this can be implemented under standard cryptographic assumptions is an interesting theoretical question, see [AHI11, CKKZ12].

Oblivious Transfer

We use different types of oblivious transfer (OT) functionalities: we denote by $\mathcal{F}_{\mathrm{OT}}$ the regular $\binom{2}{1}$-OT functionality. I.e. A inputs two *messages* m_0 and m_1 to $\mathcal{F}_{\mathrm{OT}}$ and B inputs a *choice bit* b. Nothing is output to A and m_b output to B. Generally we denote by $\binom{n}{t}$-$\mathcal{F}_{\mathrm{OT}}$ the $\binom{n}{t}$-OT functionality. Here A inputs messages $m_1, \ldots m_n$ and B inputs a *choice set* $I \subseteq [n]$ of size t. Again nothing is output to A and the messages $(m_i)_{i \in I}$ are output to B. We use the notation $\mathcal{F}_{\mathrm{OT}}(\ell, \tau)$ respectively $\binom{n}{t}$-$\mathcal{F}_{\mathrm{OT}}(\ell, \tau)$ for the OT functionalities that provide ℓ OTs with messages in $\{0,1\}^\tau$. When the length of messages and/or amount of OTs are clear from context we may drop these parameters.

We will often make use of the randomized versions of these functionalities. I.e. versions where A and B do not give any input and the functionality just picks uniformly random outputs. E.g. the randomized version of the $\mathcal{F}_{\mathrm{OT}}$

functionality samples m_0, m_1 and b uniformly at random and then outputs m_0 and m_1 to A and b and m_b to B. When considering the randomized version of OT we will let corrupted players choose their own random values. Say B above was corrupted, he then first gets to input m_b and b, the functionality then samples $m_{b\oplus 1}$ uniformly at random and outputs m_0 and m_1 to A. We denote these randomized versions of OT by ROT and name the corresponding functionalities $\mathcal{F}_{\mathrm{ROT}}$ and $\binom{n}{t}$-$\mathcal{F}_{\mathrm{ROT}}$.

We note that we can easily implement $\mathcal{F}_{\mathrm{ROT}}(\ell, \tau)$ for any $\tau = \mathrm{poly}(\psi)$ using a prg and a $\mathcal{F}_{\mathrm{ROT}}(\ell, \psi)$ functionality: simply use $\mathcal{F}_{\mathrm{ROT}}(\ell, \psi)$ to transfer seeds of length ψ and then use the prg to extend the seeds to τ bits.

Equality Check

We use a functionality $\mathcal{F}_{\mathrm{EQ}}(\ell)$ which allows two parties to check that two strings of length ℓ are equal. If they are different the functionality leaks both strings to the adversary, which makes secure implementation easier. We define and use this functionality to simplify the exposition of our protocol. In practice we implement the functionality by letting the parties compare exchanged hash's of their values: this is a secure implementation of the functionality in the random oracle model.

For completeness we give a protocol which securely implements $\mathcal{F}_{\mathrm{EQ}}$ in the RO model. Let $H : \{0,1\}^* \to \{0,1\}^\psi$ be a hash function, modeled as a RO as described above. Let ψ be the security parameter. To test strings x and y for equality A and B do the following.

1. A chooses a random string $r \in_R \{0,1\}^\psi$, sends $c = H(x||r)$ to B.
2. B sends y to A.
3. A sends x, r to B. A outputs $x \stackrel{?}{=} y$.
4. B outputs $(H(x||r) \stackrel{?}{=} c) \wedge (x \stackrel{?}{=} y)$.

This is a secure implementation of the $\mathcal{F}_{\mathrm{EQ}}(\ell)$ functionality in the RO model. If A is corrupted, the simulator extracts x, r from the simulated call to the RO, if the hash function was queried with an input which yielded the c sent by A. Then, it inputs x to $\mathcal{F}_{\mathrm{EQ}}$ and receives (x, y) from the ideal functionality (if $x \neq y$). If the hash function was not queried with an input which yielded the c sent by A, then the simulator inputs a uniformly random x to $\mathcal{F}_{\mathrm{EQ}}$ and receives (x, y). It then sends y to the corrupted A. On input x', r' from A, if $(x', r') \neq (x, r)$ the simulator inputs "abort" to the $\mathcal{F}_{\mathrm{EQ}}$ functionality on behalf of A, or "deliver" otherwise. If $(x', r') = (x, r)$, simulation is perfect. If they are different, the only way that the environment can distinguish is by finding $(x', r') \neq (x, r)$ s.t. $H(x||r) = H(x'||r')$ or by finding (x', r') such that $c = H(x'||r')$ for a c which did not result from a previous query. In the random oracle both events happen with probability less than $\mathrm{poly}(\psi)2^{-\psi}$, as the environment is only allowed a polynomial number of calls to the RO.

If B is corrupted, then the simulator sends a random value $c \in_R \{0,1\}^\ell$ to B. Then, on input y from B it inputs this value to the $\mathcal{F}_{\mathrm{EQ}}$ functionality and

receives (x, y). Now, it chooses a random $r \in_R \{0,1\}^{\psi}$ and programs the RO to output c on input $x||r$, and sends x and r to B. Simulation is perfect, and the environment can only distinguish if it had already queried the RO on input $x||r$, and this happens with probability $\text{poly}(\psi)2^{-\psi}$, as $r \in \{0,1\}^{\psi}$ is uniformly random, and the environment is only allowed a polynomial number of calls to the RO.

Part I

TinyOT

Chapter 3

Extending Oblivious Transfer

In this chapter we show how we can produce a virtually unbounded number of OTs from a small number of seed OTs. The amortized work per produced OT is linear in ψ, the security parameter, or simply a few calls to a hash function.

In fact we will concentrate on extending a few OTs to random OTs (ROTs). I.e. we will show generate a very large number of ROTs from a few OTs. As shown in [Bea95] one can easily obtain OT from precomputed ROTs, thus this is essentially with out loss of generality.

To implement our OT-extension protocol we go via yet an other variant of OT we call Δ-ROT, inspired by an intermediate step of the highly efficient semi-honest OT-extender of [IKNP03]. A Δ-ROT is a variant of ROT where the senders messages M_0 and M_1 are correlated in such a way that $M_0 \oplus M_1 = \Delta$, for some constant Δ unknown to B, which we will call the *global key*. Given a functionality that provides Δ-ROTs it is very easy to implement ROT, thus the main work of this chapter goes into extending a few OTs to many Δ-ROTs efficiently and with malicious security.

Overview

- As we shall see our construction goes via a number of leaky functionalities. I.e. functionalities where the adversary might mount an attack to gain some leakage. Therefore, we will start this chapter by introducing the idea of a *leakage agent* to abstract away the concrete attacks of the adversary.

- In Section 3.2 we implement $\mathcal{F}_{\text{ROT}}$ using a functionality providing a variant of Δ-ROT which are *leaky*. By leaky we mean that B may learn a few bits of the global key Δ. We call this functionality a leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality. The implementation simply uses a hash function (modeled as a random oracle) to break the correlation between messages from $\mathcal{F}_{\Delta\text{-ROT}}$.

- In Section 3.3 we then consider a a functionality similar to the $\mathcal{F}_{\Delta\text{-ROT}}$ functionality but with reversed roles. I.e. with A as the receiver and B as sender. We call this functionality $\mathcal{F}_{\Gamma\text{-ROT}}$. We notice that if we relax this functionality, by allowing a few of A's choice bits to leak to B the resulting *leaky* $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality is essentially equivalent to the leaky

$\mathcal{F}_{\Delta\text{-ROT}}$ functionality. I.e. the leakage on choice bits in $\mathcal{F}_{\Gamma\text{-ROT}}$ becomes leakage on Δ in $\mathcal{F}_{\Delta\text{-ROT}}$.

- In Section 3.4 we implement the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality using a few OT's in the following way: A inputs random choice bits c_i in an OT, and B is supposed to input random messages $(N_0^i, N_0^i \oplus \Gamma)$ so that A receives $N_{c_i}^i = N_0^i \oplus c_i\Gamma$. To ensure that B is being honest and uses the same Γ in most OTs a check is performed, which restricts B to cheat in at most a few OTs. We notice that what B gains by using inconsistent Γ's in a few OTs is no more than learning a few of A's choice bits c_i, thus implementing the $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality.

- While the leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality is all we need for our OT-extension protocol, in Chapter 4 it is useful to have a non-leaky version of this functionality. Therefore, in Section 3.6, using privacy amplification, we use the leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality to implement a $\mathcal{F}_{\Delta\text{-ROT}}$ functionality where the Δ value is shorter than in the leaky functionality but fully secure.

- Finally in Section 3.7 we sketch a complexity analysis counting the symmetric primitives used in the protocol.

3.1 Leakage Agent

As described above, our construction will go via a number of functionalities that are leaky in some way. The setting is that there is a secret string $\Delta \in_{\mathrm{R}} \{0,1\}^\tau$ and an adversary A who tries to guess Δ. A can launch an attack which might leak some of the bits of Δ, but with some probability A's attack will be detected. In this section we introduce the idea of a *leakage agent*, in order to abstract away the concrete leakage attacks.

A leakage agent LA will be an interactive PPT algorithm that will interact with A and then possibly specify what leakage to give to A: A gets to first interact with LA in some way (specified by the interface of LA). Following this interaction we input $\Delta \in_{\mathrm{R}} \{0,1\}^\tau$ to LA. Based on the interaction with A LA computes some $S \subseteq [\tau]$ and $c \in \{0,1\}$ and outputs (S, c). Here S indicates the bits of Δ to be leaked to A if the attack is undetected, and c whether or not the attack is detected ($c = 0$ indicating detection). This models that during A's attack, i.e. the interaction with LA, no bits of Δ leak, but after the attack the set of bits that does leak may depend on Δ it self.

We need a measure of how many bits a leakage agent LA leaks. We do this via a game against an unbounded adversary A.

$\underline{\mathrm{LeakageGame}(\mathsf{LA}, \mathsf{A}, \tau)}$
$\Delta \in_{\mathrm{R}} \{0,1\}^\tau$
$\mathsf{A}(\tau) \leftrightarrow \mathsf{LA}(\tau)$
$(S, c) \leftarrow \mathsf{LA}(\Delta)$
$(g_i)_{i \in \bar{S}} \leftarrow \mathsf{A}(S, (\Delta_i)_{i \in S})$

Where $\bar{S} = [\tau] \setminus S$ and $\mathsf{A}(\tau) \leftrightarrow \mathsf{LA}(\tau)$ means A and LA interact each given τ as input.

We say that A wins if $c = 1$ and $(\Delta_i)_{i \in \bar{S}} = (g_i)_{i \in \bar{S}}$. Furthermore, we say that an adversary A is *optimal* if she has the highest probability of any adversary of winning LeakageGame$(\mathsf{LA}, \mathsf{A}, \tau)$. If there were no leakage, i.e., $S = \emptyset$, then it is clear that an optimal A wins the game with probability exactly $2^{-\tau}$. If A is always given exactly s bits and is never detected, then it is clear that an optimal A can win the game with probability exactly $2^{s-\tau}$. This motivates defining the number of bits leaked by LA to be $\mathrm{leak}_{\mathsf{LA}} \stackrel{\text{def}}{=} \log_2(\mathrm{success}_{\mathsf{LA}}) + \tau$, where $\mathrm{success}_{\mathsf{LA}}$ is the probability that an optimal A wins the leakage game.

We say that LA is κ-secure if $\tau - \mathrm{leak}_{\mathsf{LA}} \geq \kappa$, and if LA is κ-secure, then no A can win the game with probability better than $2^{-\kappa}$.

We now rewrite the definition of $\mathrm{leak}_{\mathsf{LA}}$ to make it more workable. We denote by $(\mathsf{LA}, \mathsf{A})(\tau)$ the experiment sampling (S, c) as in LeakageGame$(\mathsf{LA}, \mathsf{A}, \tau)$. It is clear that an optimal A can guess all Δ_i for $i \in \bar{S}$ with probability exactly $2^{|S|-\tau}$. This means that an optimal A wins with probability

$$\sum_{s=0}^{\tau} \Pr\left((S, c) \leftarrow (\mathsf{LA}, \mathsf{A})(\tau) : |S| = s \wedge c = 1\right) 2^{s-\tau} \ .$$

To simplify this expression we define index variables $I_s, J_s \in \{0, 1\}$ where I_s is 1 iff $c = 1$ and $|S| = s$ and J_s is 1 iff $|S| = s$ when $(S, c) \leftarrow (\mathsf{LA}, \mathsf{A})(\tau)$. Note that $I_s = cJ_s$ and that $\sum_s J_s 2^s = 2^{|S|}$. So, if we take expectation over $(S, c) \leftarrow (\mathsf{LA}, \mathsf{A})(\tau)$, then we get that

$$\sum_{s=0}^{\tau} \Pr\left((S, c) \leftarrow (\mathsf{LA}, \mathsf{A})(\tau) : |S| = s \wedge c = 1\right) 2^s = \sum_{s=0}^{\tau} \mathrm{E}\left[I_s\right] 2^s \ ,$$

where

$$\begin{aligned}
\sum_{s=0}^{\tau} \mathrm{E}\left[I_s\right] 2^s &= \mathrm{E}\left[\sum_{s=0}^{\tau} I_s 2^s\right] \\
&= \mathrm{E}\left[\sum_{s=0}^{\tau} c J_s 2^s\right] \\
&= \mathrm{E}\left[c \sum_{s=0}^{\tau} J_s 2^s\right] \\
&= \mathrm{E}\left[c 2^{|S|}\right] \ .
\end{aligned}$$

Hence $\mathrm{success}_{\mathsf{LA}} = \max_{\mathsf{A}}(2^{-\tau} \mathrm{E}\left[c 2^{|S|}\right])$ and

$$\log_2(\mathrm{success}_{\mathsf{LA}}) = -\tau + \log_2 \max_{\mathsf{A}} \left(\mathrm{E}\left[c 2^{|S|}\right]\right) \ ,$$

which shows that

$$\mathrm{leak}_{\mathsf{LA}} = \max_{\mathsf{A}} \log_2 \left(\mathrm{E}\left[c 2^{|S|}\right]\right) \ .$$

Honest Parties

On input `start` from both A and B the functionality does the following.

1. The functionality samples $(X_0^i, X_1^i)_{i\in[\ell]} \in_R \{0,1\}^{2\psi\ell}$, and outputs (X_0^i, X_1^i) to A.
2. The functionality samples $(b_i)_{i\in[\ell]} \in_R \{0,1\}^{\ell}$ and outputs $(X_{b_i}^i, b_i)_{i\in[\ell]}$ to B.

Corrupt Parties

1. If A is corrupted the functionality waits for her to input $(X_0^i, X_1^i)_{i\in[\ell]} \in \{0,1\}^{2\psi\ell}$. The functionality then outputs as above using these values.
2. If B is corrupted the functionality waits for him to input $(X_{b_i}^i, b_i)_{i\in[\ell]} \in \{0,1\}^{\ell(\psi+1)}$. The functionality then outputs as above using these values.

Figure 3.1: The Random OT functionality $\mathcal{F}_{\text{ROT}}(\ell, \psi)$

3.2 $\mathcal{F}_{\text{ROT}}$ from $\mathcal{F}_{\Delta\text{-ROT}}$

In this section we show how we implement the $\mathcal{F}_{\text{ROT}}$ as described in Figure 3.1.

To this end we introduce the notion of a Δ-*ROT*. In contrast to ROT, in Δ-ROT the sender A first receives random *global key* Δ and then for each Δ-ROT A receives only one random message M_0^i. The *other* message is defined to be $M_1^i = M_0^i \oplus \Delta$. I.e. B receives random choice bit b_i and message $M_{b_i}^i = M_0^i \oplus b_i\Delta$. To implement ROT it will suffice for us to consider a *leaky* variant of Δ-ROT. Namely, a variant where B may learn some bits of the global key Δ. We call the functionality that provides such leaky Δ-ROTs a LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality and describe it in detail Figure 3.2, where LA is a leakage agent as described above. As long as the leakage agent LA is ψ-secure such a functionality turns out to be enough to implement ROT.

To implement $\mathcal{F}_{\text{ROT}}$ using LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}$, we notice that the $\mathcal{F}_{\Delta\text{-ROT}}$ functionality resembles an intermediate step of the passive-secure OT extension protocol of [IKNP03]: $\mathcal{F}_{\Delta\text{-ROT}}$ is a random OT, where all the sender's messages are correlated, so that the XOR of the messages in any OT is a constant (the global key of the $\mathcal{F}_{\Delta\text{-ROT}}$). This correlation can be easily broken using the random oracle. This idea leads to the protocol for $\mathcal{F}_{\text{ROT}}$ described in Figure 3.3.

Theorem 3.1. *Let ψ be the security parameter. The protocol in Figure 3.3 securely implements $\mathcal{F}_{\text{ROT}}(\ell, \psi)$ in the LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \tau)$-hybrid model, where LA is ψ-secure on τ bits. The work is $O(\tau\ell)$*[1]*.*

Proof. Correctness is simple: we have that $M_{b_i}^i = M_0^i \oplus b_i\Delta$, so $Y^i = H(M_{b_i}^i) = H(M_0^i \oplus b_i\Delta) = X_{b_i}^i$. Clearly the protocol leaks no information on the b_i as there is no communication from B to A. It is therefore sufficient to look at the case of a corrupted B^*. We are not going to give a simulation argument but just show that $X_{1\oplus b_i}^i$ is uniformly random to B^* except with probability $\text{poly}(\psi)2^{-\psi}$.

Since $X_{1\oplus b_i}^i = H(M_0^i \oplus (1 \oplus b_i)\Delta)$ and H is a random oracle, it is clear that $X_{1\oplus b_i}^i$ is uniformly random to B^* until B^* queries H on $Q = M_0^i \oplus (1 \oplus b_i)\Delta$.

[1] counting hashing of τ bits as $O(\tau)$ work.

Honest Parties

On input `start` from both A and B the functionality does the following.

1. The functionality samples $\Delta \in_R \{0,1\}^\tau$ and outputs it to A.
2. For all $i \in [\ell]$ the functionality samples $b_i \in_R \{0,1\}$ and $M_0^i \in_R \{0,1\}^\tau$.
3. The functionality outputs $(M_0^i \oplus b_i\Delta, b_i)_{i\in[\ell]}$ to B and $(M_0^i)_{i\in[\ell]}$ to A.

Corrupted Parties

1. If B is corrupted the functionality runs LA to sample (S, c), with B playing the role of the adversary and where the functionality inputs Δ to LA as its secret string. If $c = 0$ the functionality outputs `fail` to A and terminates. Otherwise, the functionality outputs $(i, \Delta_i)_{i\in S}$ to B.
2. Furthermore, if B is corrupted, the functionality waits to give output till it receives the message $(\hat{M}^i_{\hat{b}_i}, \hat{b}_i)_{i\in[\ell]}$ from B, where $\hat{M}^i_{\hat{b}_i} \in \{0,1\}^\tau$ and $\hat{b}_i \in \{0,1\}$. The functionality then sets $b_i = \hat{b}_i$ and $M_0^i = \hat{M}^i_{b_i} \oplus b_i\Delta$ and outputs as described above.
3. If A is corrupted, the functionality waits to give output till it receives the message $(\hat{\Delta}, (\hat{M}_0^i)_{i\in[\ell]})$ from A, where $\hat{\Delta}, \hat{M}_0^i \in \{0,1\}^\tau$. The functionality then sets $\Delta = \hat{\Delta}$ and $M_0^i = \hat{M}_0^i$ and outputs as described above.

Figure 3.2: The LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \tau)$ functionality

1. A and B call a LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \tau)$ functionality. The output to B is $(M^i_{b_i}, b_i)_{i\in[\ell]}$. The output to A is $(\Delta, (M_0^i)_{i\in[\ell]})$.
2. B computes $Y^i = H(M^i_{b_i}) \in \{0,1\}^\psi$ and outputs $(Y^i, b_i)_{i\in[\ell]}$.
3. A computes $X_0^i = H(M_0^i) \in \{0,1\}^\psi$ and $X_1^i = H(M_0^i \oplus \Delta) \in \{0,1\}^\psi$ and outputs $(X_0^i, X_1^i)_{i\in[\ell]}$.

Figure 3.3: The protocol for reducing $\mathcal{F}_{\text{ROT}}(\ell, \psi)$ to LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \psi)$

Since $M^i_{b_i} = M_0^i \oplus b_i\Delta$ we have that $Q = M_0^i \oplus (1 \oplus b_i)\Delta$ would imply that $M^i_{b_i} \oplus Q = \Delta$. So, if we let B^* query H, say, on $Q \oplus M^i_{b_i}$ each time it queries H on some Q, which would not change its asymptotic running time, then we have that all $X^i_{1\oplus b_i}$ are uniformly random to B^* until it queries H on Δ.

However this happens with probability at most $2^{-\psi}$ by the ψ-security of LA. Namely notice that all inputs to B^* are independent of Δ except for what leakage he may get from LA. Thus, a B^* that queries H on Δ with probability better than $2^{-\psi}$ would contradict the ψ-security of LA. □

3.3 Leaky $\mathcal{F}_{\Delta\text{-ROT}}$ From Leaky $\mathcal{F}_{\Gamma\text{-ROT}}$

We now consider a functionality similar to the leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality, but reversed in the sense that A plays the role of receiver and B the role of sender and that the leakage is on the choice bits instead of the global key. We call this functionality a leaky $\mathcal{F}_{\Gamma\text{-ROT}}$, and formally describe it in Figure 3.4.

The LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ provides τ leaky Γ-ROTs with messages of length ℓ. It turns out that the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality is actually equivalent to the leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality. I.e. given one functionality simply renaming the

Honest Parties

On input `start` from both A and B the functionality does the following.

1. The functionality samples $\Gamma \in_R \{0,1\}^\ell$ and outputs it to B.
2. For all $i \in [\tau]$ the functionality samples $c_i \in_R \{0,1\}$ and $N_0^i \in_R \{0,1\}^\tau$.
3. The functionality outputs $(N_0^i \oplus c_i\Gamma, c_i)_{i\in[\tau]}$ to A and $(N_0^i)_{i\in[\tau]}$ to B.

Corrupted Parties

1. If B is corrupted the functionality runs LA to sample (S, c), with B playing the role of the adversary and where the functionality inputs $(c_i)_{i\in[\tau]}$ to LA as its secret string. If $c = 0$ the functionality outputs `fail` to B and terminates. Otherwise, the functionality outputs $(i, c_i)_{i\in S}$ to B.
2. As in the Figure 3.2 any corrupted player is allowed to specify the value of its output.

Figure 3.4: The LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ functionality

outputs of gives us the other, as demonstrated in Figure 3.5. We say this more formally in Theorem 3.2.

1. A and B invoke a LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ functionality. A learns $(N_{c_i}^i, c_i)_{i\in[\tau]}$ and B learns $(\Gamma, (N_0^i)_{i\in[\tau]})$.
2. B lets b_j be the j'th bit of Γ and $M_{b_j}^j$ the string consisting of the j'th bits from all the strings N_0^i, i.e. $M_{b_j}^j = N_{0,j}^1 || N_{0,j}^2 || \dots || N_{0,j}^\ell$.
3. A lets Δ be the string consisting of all the bits c_i, i.e. $\Delta = c_1||c_2||\dots||c_\tau$, and lets M_0^j be the string consisting of the j-th bits from all the strings $N_{c_i}^i$, i.e. $M_0^j = N_{c_1,j}^1 || N_{c_2,j}^2 || \dots || N_{c_\tau,j}^\tau$. [a]

[a] It may be easier to think of this renaming in terms of bit matrices as in (3.1)

Figure 3.5: Protocol reducing LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \tau)$ to LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$

Theorem 3.2. *For all ℓ, τ and LA the LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \tau)$ and $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ functionalities are linear locally equivalent, i.e., can be implemented given the other in linear time without interaction.*

Proof. The first direction (reducing leaky $\mathcal{F}_{\Delta\text{-ROT}}$ to $\mathcal{F}_{\Gamma\text{-ROT}}$) is shown in Figure 3.5. The other direction ($\mathcal{F}_{\Gamma\text{-ROT}}$ to $\mathcal{F}_{\Delta\text{-ROT}}$) will follow by the fact that the renamings are reversible in linear time. One can easily verify that for all $j \in [\ell]$, $M_{b_j}^j$ is the correct message given choice bit b_j and the values M_0^j and Δ, i.e. $M_{b_j}^j = M_0^j \oplus b_j\Delta$. This is perhaps easiest seen by viewing the renaming described in Figure 3.5 in terms of bit matrices where the strings $N_c^i, \Gamma \in \{0,1\}^\ell$ are viewed as column vectors and strings $M_b^j, \Delta \in \{0,1\}^\tau$ are viewed as row vectors. Taking this view the following holds

$$\begin{pmatrix} | & & | \\ N_{c_1}^1 & \dots & N_{c_\tau}^\tau \\ | & & | \end{pmatrix} = \begin{pmatrix} | & & | \\ N_0^1 & \dots & N_0^\tau \\ | & & | \end{pmatrix} \oplus \begin{pmatrix} | & & | \\ c_1\Gamma & \dots & c_\tau\Gamma \\ | & & | \end{pmatrix}$$
$$= \begin{pmatrix} — & M_{b_1}^1 & — \\ & \vdots & \\ — & M_{b_\ell}^\ell & — \end{pmatrix} \oplus \begin{pmatrix} — & b_1\Delta & — \\ & \vdots & \\ — & b_\ell\Delta & — \end{pmatrix} \tag{3.1}$$
$$= \begin{pmatrix} — & M_0^1 & — \\ & \vdots & \\ — & M_0^\ell & — \end{pmatrix} \in \{0,1\}^{\ell\times\tau},$$

where the first equality is by definition of $\mathcal{F}_{\Gamma\text{-ROT}}$ and the second is by design of the protocol in Figure 3.5.

It is easy to verify (as the protocol only consists of renamings) that leakage on the choice bits c_i is equivalent to leakage on the global key Δ under this transformation. Giving a simulation argument is then straight forward when LA is the same for both functionalities. □

Note that doing this simple renaming we turn a LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}(\ell,\tau)$ functionality into a LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\tau,\ell)$ functionality. If we choose $\ell = \text{poly}(\psi)$ this means that we can turn τ leaky Γ-ROTs into a very larger number (ℓ) of leaky Δ-ROTs. I.e. implementing the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality for a ψ-secure leakage agent LA on $\tau = O(\psi)$ bits, using only $\mathcal{F}_{\text{OT}}(\tau,\ell)$ we have our OT-extension protocol. In the following section we give such an implementation of the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality.

3.4 A Protocol For leaky $\mathcal{F}_{\Gamma\text{-ROT}}$

In this section we show how to construct leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ from $\mathcal{F}_{\text{OT}}$. The protocol ensures that most of the choice bits are kept secret.

The main idea of the protocol, described in Figure 3.6, is the following: B and A run many OTs using an $\mathcal{F}_{\text{OT}}$ functionality. In the i'th OT B inputs messages N_0^i and N_1^i and A inputs choice bit c_i. An honest B sets her messages so that $N_0^i \oplus N_1^i = \Gamma$ for all i. To test that B used the same value for Γ in every OT the parties randomly partition the OTs into pairs. Say that one such pair consists of the i'th and j'th OT. A then sends $d = c_i \oplus c_j$ to B and computes $D = N_{c_i}^i \oplus N_{c_j}^i$. If B is honest she can also compute D as $D = N_0^i \oplus N_0^i \oplus d\Gamma$. On the other hand if she used different values for Γ in the i'th and j'th OT she can guess D with at most probability $\frac{1}{2}$, as we shall demonstrate below. Therefore, to test that B behaved honestly the parties can use the $\mathcal{F}_{\text{EQ}}$ functionality to test that they are both able to compute D. As A reveals $c_i \oplus c_j$, they waste the j'th OT and only use the output of the i'th OT as output from the protocol—as c_j is uniformly random $c_i \oplus c_j$ leaks no information on c_i. Note that we cannot simply let B reveal the D, as a malicious A could send $d = 1 \oplus c_i \oplus c_j$: this

would allow A to learn both D and $D \oplus \Gamma$, thus leaking Γ. Using $\mathcal{F}_{\text{EQ}}$ forces a A who uses this attack to guess a random message he did not see, which he can do only with negligible probability $2^{-\ell}$.

1. B samples $\Gamma \in_{\text{R}} \{0,1\}^{\ell}$ and for $i = 1, \ldots, 2\tau$ samples $N_0^i \in_{\text{R}} \{0,1\}^{\ell}$.
2. A samples $(c_1, \ldots, c_{2\tau}) \in_{\text{R}} \{0,1\}^{2\tau}$.
3. The parties run a $\mathcal{F}_{\text{OT}}(2\tau, \ell)$ functionality, where for $i = 1, \ldots, 2\tau$ B inputs messages N_0^i and $N_0^i \oplus \Gamma$. A inputs choice bit c_i and receives $N_{c_i}^i = N_0^i \oplus c_i\Gamma$.
4. A picks a uniformly random pairing π (a permutation $\pi : [2\tau] \to [2\tau]$ where $\forall i, \pi(\pi(i)) = i$), and sends π to B. Given a pairing π, let $\mathcal{S}_\pi = \{i | i \leq \pi(i)\}$, i.e., for each pair, add the smallest index to $\mathcal{S}_\pi$.
5. For all τ indices $i \in \mathcal{S}_\pi$:
   (a) A announces $d_i = c_i \oplus c_{\pi(i)}$.
   (b) A computes $D_i = N_{c_i}^i \oplus N_{c_j}^j$ and B computes $\hat{D}_i = N_0^i \oplus N_0^j \oplus d_i\Gamma$.

   The parties then compare the strings $(D_i)_{i \in \mathcal{S}_\pi}$ and $(\hat{D}_i)_{i \in \mathcal{S}_\pi}$ using $\mathcal{F}_{\text{EQ}}(\tau\ell)$ and abort if they are different. Otherwise the protocol continues.
6. B outputs $(\Gamma, (N_0^i)_{i \in \mathcal{S}_\pi})$ and A outputs $(N_{c_i}^i, c_i)_{i \in \mathcal{S}_\pi}$.

Figure 3.6: The protocol for reducing $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ to $\mathcal{F}_{\text{OT}}(2\tau, \ell)$ and $\mathcal{F}_{\text{EQ}}(\tau\ell)$.

Theorem 3.3 tell us that the protocol in Figure 3.6. The proving this will take a lot of work and therefore, we push the proof to Section 3.5 and immediately present Cor. 3.2 summing up the results of this chapter so far.

Theorem 3.3. *Let $\kappa = \frac{3}{4}\tau$, and let* LA *be a κ-secure leakage agent on τ bits. The protocol in Figure 3.6 securely implements* LA*-leaky $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ in the $(\mathcal{F}_{\text{OT}}(2\tau, \ell), \mathcal{F}_{\text{EQ}}(\tau\ell))$-hybrid model. The communication is $O(\tau)$. The work is $O(\tau\ell)$.*

Corrollary 3.1. *Let ψ denote the security parameter and let $\ell = \text{poly}(\psi)$. The functionality $\mathcal{F}_{\text{ROT}}(\ell, \psi)$ can be reduced to $(\mathcal{F}_{\text{OT}}(\frac{8}{3}\psi, \psi), \mathcal{F}_{\text{EQ}}(\psi))$. The communication is $O(\ell\psi)$ and the work is $O(\psi\ell)$.*

Proof. Combining Theorem 3.1, 3.2 and 3.3 and setting $\tau = \frac{4}{3}\psi$ we have that $\mathcal{F}_{\text{ROT}}(\ell, \psi)$ can be reduced to $(\mathcal{F}_{\text{OT}}(\frac{8}{3}\psi, \ell), \mathcal{F}_{\text{EQ}}(\frac{4}{3}\psi\ell))$ with communication $O(\ell\psi)$ and work $O(\psi\ell)$. For any polynomial ℓ, we can implement $\mathcal{F}_{\text{OT}}(\frac{8}{3}\psi, \ell)$ given $\mathcal{F}_{\text{OT}}(\frac{8}{3}\psi, \psi)$ and a pseudo-random generator prg : $\{0,1\}^{\psi} \to \{0,1\}^{\ell}$. Namely, seeds are sent using the OTs and the prg is used to one-time pad encrypt the messages. The communication is 2ℓ. If we use the RO to implement the pseudo-random generator and count the hashing of ψ bits as $O(\psi)$ work, then the work is $O(\ell\psi)$ (using ℓ/ψ calls to H to expand ψ bit seeds to ℓ bits). We can implement $\mathcal{F}_{\text{EQ}}(\frac{4}{3}\psi\ell)$ by comparing short hashes produced using the RO. The work is $O(\psi\ell)$ (using $\frac{4}{3}\ell$ calls to H to hash the $\frac{4}{3}\psi\ell$-bit string down to ψ-bits).

□

Since the oracles $(\mathcal{F}_{\text{OT}}(\frac{8}{3}\psi, \psi), \mathcal{F}_{\text{EQ}}(\psi))$ are independent of ℓ, the cost of essentially any reasonable implementation of them can be amortized away by picking ℓ large enough.

3.5 Proof Of Theorem 3.3

In this section we give the full proof of Theorem 3.3. The cases where no party is corrupted and where A is corrupted is straight forward, so we will focus on the case of corrupted B^*.

The proof goes via a two intermediary functionalities, each modelling a different aspect of the intuition given in Section 3.4. Between these intermediate functionality we show linear reducibility. We will then construct a specific leakage agent LA so that the LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ is linearly reducible to the intermediate functionalities. Finally the proof concludes by showing that this leakage agent LA is κ-secure. Before we get into the proof we will present convenient way to model the protocol of Figure 3.6.

Modeling The Protocol

It will be helpful to model the protocol in Figure 3.6 in terms of balls colored by B^*. More specifically with each pair of messages N_0^i and N_1^i input to $\mathcal{F}_{\text{OT}}$ by B^* we associate a *ball* i. If $N_0^i \oplus N_1^i = \Gamma_i$, then we associate with Γ_i a *color* in $[2\tau]$, say j, and say that B^* colors ball i with the color j. Since there is 2τ message pair (N_0^i, N_1^i) in the protocol there can be at most 2τ balls of 2τ different colors. We will denote by col the *coloring function* $\text{col} : [2\tau] \to [2\tau]$ that maps each ball to its color. An honest B picks a single Γ and for all $i \in [2\tau]$ picks messages so that $N_0^i \oplus N_1^i = \Gamma_i = \Gamma$. I.e. an honest B will color all balls with the same color, so in this case col is a constant function. A corrupted B^*, on the other hand, can use any coloring function col.

Let $\text{col}_0, \ldots, \text{col}_{2\tau-1}$ be the possible colors, then for a coloring function col we will let C_i be the set of balls of color col_i, i.e. $C_i = \{j \in [2\tau] | \, \text{col}(j) = \text{col}_i\}$. Without loss of generality we will let col_0 be *most common color* of col, i.e. $|C_0| \geq |C_i|$ for all $i \in [2\tau]$. We will sometimes refer to the value Γ associated with col_0 as the *right value* and all other $\Gamma' \neq \Gamma$ as being *wrong*.

The pairing π used in the protocol defines pairs of balls. That is, each ball $j \in [2\tau]$ is paired with $\pi(j)$. If we let $\mathcal{S} = \mathcal{S}_\pi = \{i | i \leq \pi(i)\}$ as in Figure 3.6 we can view each $i \in \mathcal{S}$ as representing the pair of balls $(i, \pi(i))$. Furthermore, we will define the set $\mathcal{M} = \mathcal{M}_{\pi,\text{col}} \subseteq \mathcal{S}$ to be the set representing *mismatched* pairs, i.e. $\mathcal{M} = \{i \in \mathcal{S} | \, \text{col}(i) \neq \text{col}(\pi(i))\}$. In the protocol $i \in \mathcal{M}$ corresponds to the situation where for the two messages pairs (N_0^i, N_1^i) and $(N_0^{\pi(i)}, N_1^{\pi(i)})$ we have

$$N_0^i \oplus N_1^i \neq N_0^{\pi(i)} \oplus N_1^{\pi(i)} \ .$$

It will also be useful to define the set $\mathcal{N} = \mathcal{N}_{\pi,\text{col}}$ to be the subset of *matched* pairs in $\mathcal{S} \setminus \mathcal{M}$ which are not of color col_0, i.e. $\mathcal{N} = \{i \in \mathcal{S} \setminus \mathcal{M} | \, \text{col}(i) \neq \text{col}_0\}$. In the protocol $i \in \mathcal{N}$ corresponds to the situation where for the two messages pairs (N_0^i, N_1^i) and $(N_0^{\pi(i)}, N_1^{\pi(i)})$ we have

$$N_0^i \oplus N_1^i = N_0^{\pi(i)} \oplus N_1^{\pi(i)} = \Gamma' \ ,$$

but Γ' is not the value associated with col_0.

In other words $\mathcal{M}$ and $\mathcal{N}$ are the pairs where B^* deviated from the protocol. For each $i \in \mathcal{N}$ B^* will not get caught, while for $i \in \mathcal{M}$ B^* will get caught with probability $\frac{1}{2}$ as we shall show below.

Intermediate Functionality 1

We now present our first intermediate functionality $\mathcal{F}_{\text{IB1}}$. This functionality captures the idea that a corrupted B^* can only get away with using a few different values of Γ.

To see this let $i \in S_\pi$ and $j = \pi(i)$ and note that if B^* chose the two message pairs (N_0^i, N_1^i) and (N_0^j, N_1^j) so that

$$\Gamma_i = N_0^i \oplus N_1^i \neq N_0^{\pi(i)} \oplus N_1^{\pi(i)} = \Gamma_j \ ,$$

i.e. $i \in \mathcal{M}$, then A computes

$$\begin{aligned} D_i &= N_{c_i}^i \oplus N_{c_j}^j \\ &= (N_0^i \oplus c_i\Gamma_i) \oplus (N_0^j \oplus c_j\Gamma_j) \\ &= (N_0^i \oplus N_0^j) \oplus (c_i \oplus c_j)\Gamma_j \oplus c_i(\Gamma_i \oplus \Gamma_j) \\ &= (N_0^i \oplus N_0^j) \oplus d_i\Gamma_j \oplus c_i(\Gamma_i \oplus \Gamma_j) \end{aligned}$$

Since $(\Gamma_i \oplus \Gamma_j) \neq 0^\ell$ and $c_i \oplus c_j$ is fixed by announcing d_i, guessing this D_i is equivalent to guessing c_i. As B^* only knows $N_0^i, N_0^j, \Gamma_i, \Gamma_j$ and d_i, all of which are independent of c_i, she can guess c_i with probability at most $\frac{1}{2}$. If B^* cheats and uses many different values of Γ, then with high probability the pairing π will be such that there are many pairs where $\Gamma_i \neq \Gamma_\pi(i)$, and B^* will get caught with high probability. However, if she uses only few values of Γ she might pass the test.

This corresponds to saying that for B^* to get away with using coloring function col, she must guess c_i for all $i \in \mathcal{M}$. I.e. B^* gets away with using col with probability at most $2^{-|\mathcal{M}|}$. Therefore, she is not likely to get away with using a coloring function that colors the balls many different colors, because such a coloring will result in $|\mathcal{M}|$ being large with high probability.

Lemma 3.1. *The protocol in Figure 3.6 implements $\mathcal{F}_{\text{IB1}}$ in the $(\mathcal{F}_{\text{OT}}(2\tau, \ell), \mathcal{F}_{\text{EQ}}(\tau\ell))$-hybrid model. I.e. $\mathcal{F}_{\text{IB1}}$ is linear reducible to $(\mathcal{F}_{\text{OT}}(2\tau, \ell), \mathcal{F}_{\text{EQ}}(\tau\ell))$.*

Proof. By observing B^*'s inputs to the OTs, the simulator learns all (N_0^i, N_1^i). Let $L_i = N_0^i$ and $\Gamma_i = N_0^i \oplus N_1^i$.

Let f be the number of distinct values in $(\Gamma_i)_{i=1}^{2\tau}$ and pick distinct $\Lambda_1, \ldots, \Lambda_f$ and $\text{col} : [2\tau] \to [2\tau]$ so that $\Gamma_i = \Lambda_{\text{col}(i)}$. For $i = f+1, \ldots, 2\tau$ pick the remaining $\Lambda_i \in \{0,1\}^\ell$ in any arbitrary way (these will not be used anyway). By construction

$$\begin{aligned} N_1^i &= L_i \oplus \Gamma_i \\ &= L_i \oplus \Lambda_{\text{col}(i)} \ . \end{aligned}$$

Honest-Parties

As in the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality.

Corrupted Parties

1. If A is corrupted: As in the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality.
2. (a) If B^* is corrupted, the functionality waits to give output till B^* inputs $(\texttt{colors}, \text{col}, (\Lambda_i, L_i)_{i\in[2\tau]})$, where $L_i, \Lambda_i \in \{0,1\}^\ell$ and col is a coloring function.
   (b) Then the functionality samples a uniformly random pairing $\pi : [2\tau] \to [2\tau]$ and outputs $(\texttt{pairs}, \pi)$ to B^*. Let $\mathcal{S} = \mathcal{S}_\pi$ and let $\mathcal{M} = \mathcal{M}_{\pi,col}$.
   (c) The functionality then waits for B^* to input $(\texttt{guess}, (g_i)_{i\in\mathcal{M}})$.
   (d) The functionality samples $(c_i)_{i\in[2\tau]} \in_R \{0,1\}^{2\tau}$. Then the functionality lets $c = 1$ if $g_i = c_i$ for all $i \in \mathcal{M}$, otherwise it lets $c = 0$. If $c = 0$ the functionality outputs `fail` to A and terminates. Otherwise, for $i \in \mathcal{S}$ it computes $N^i_{c_i} = L_i \oplus c_i\Lambda_{\text{col}(i)}$ and outputs $(N^i_{c_i}, c_i)_{i\in\mathcal{S}}$ to A.

Figure 3.7: The First Intermediate Functionality $\mathcal{F}_{\text{IB1}}$

Input $(\texttt{colors}, \text{col}, (\Lambda_i, L_i)_{i\in[2\tau]})$ to $\mathcal{F}_{\text{IB1}}$ on behalf of B^* and receive $(\texttt{pairs}, \pi)$. Send π to B^* as if coming from A along with uniformly random $\{d_i\}_{i\in\mathcal{S}}$.

Then observe the inputs $\hat{D}_i$ from B^* to the $\mathcal{F}_{\text{EQ}}$ functionality. The simulator must now pick the guesses g_i for $i \in \mathcal{M}$. Note that $i \in \mathcal{M}$ implies that $\Lambda_{\text{col}(i)} \neq \Lambda_{\text{col}(\pi(i))}$, which means that $\Gamma_i \neq \Gamma_{\pi(i)}$. We use this to pick g_i, as follows: after seeing d_i, B^* knows that $c_i = d_i \oplus c_{\pi(i)}$. Hence an honest A would input to the comparison the following value for D_i depending on c_i

$$D_i(c_i) = (L_i \oplus L_{\pi(i)}) \oplus d_i\Lambda_{\text{col}(\pi(i))} \oplus c_i(\Lambda_{\text{col}(i)} \oplus \Lambda_{\text{col}(\pi(i))}) \ .$$

As $\Lambda_{\text{col}(i)} \neq \Lambda_{\text{col}(\pi(i))}$ we have $D_i(0) \neq D_i(1)$. Thus if B^*'s input to the $\mathcal{F}_{\text{EQ}}$ functionality $\hat{D}_i$ is equal to $D_i(0)$ (resp. $D_i(1)$), the simulator sets $g_i = 0$ (resp. $g_i = 1$). If the simulator is able to set all $(g_i)_{i\in[\tau]}$ in this manner, i.e. all $\hat{D}_i$ equal either $D_i(0)$ or $D_i(1)$, it inputs $(\texttt{guess}, (g_i)_{i\in[\tau]})$ to the $\mathcal{F}_(IB1)$ functionality. Otherwise, the simulator outputs `fail` and aborts.

Notice that in the real world protocol, if $g_i = c_i$, then $D_i = D_i(g_i) = \hat{D}_i$ and B^* passes the test. If $g_i \neq c_i$, then $D_i = D_i(1 \oplus g_i) \neq \hat{D}_i$ and B^* fails the test. So, the protocol and the simulation fails on the same event. Note then that when the functionality does not fail, then it outputs

$$\begin{aligned} L_i \oplus c_i\Lambda_{\text{col}(i)} &= N^i_0 \oplus c_i\Gamma_i \\ &= N^i_0 \oplus c_i(N^i_0 \oplus N^i_1) \\ &= N^i_{c_i} \ , \end{aligned}$$

exactly as the protocol. Hence the simulation is perfect.

□

Intermediate functionality 2

The second intermediate functionality $\mathcal{F}_{\text{IB2}}$ in Figure 3.8 captures the idea that an adversary that gets away with using multiple values of Γ is equivalent to one that only uses a single value of Γ but instead learns some of A's choice bits c_i.

To see this first recall that we above showed that for B^* to get away with using multiple values of Γ she must guess all c_i for $i \in \mathcal{M}$. Doing so confirms her guess on c_i, so if she passes the test she learns these c_i. Now we can let Γ be the value associated with the most common color col_0. Assume then that B^* cheated and for some i (not necessarily in $\mathcal{M}$) used a message pair (N_0^i, N_1^i) where $N_0^i \oplus N_1^i = \Gamma' \neq \Gamma$. We can explain this as an honest run: If $c_i = 0$, the run is equivalent to B^* having inputted $(N_0^i, N_0^i \oplus \Gamma)$, as A gets no information on the second message when $c_i = 0$ (by privacy of $\mathcal{F}_{\text{OT}}$). If $c_i = 1$, then the run is equivalent to having input message $(N_1^i \oplus \Gamma, N_1^i)$ as A gets no information on the first message when $c_i = 1$. So, any cheating strategy of B^* can be simulated by letting her honestly use the same Γ in all message pairs and then let her try to guess the bits c_i for $i \in \mathcal{M}$. If her guess is incorrect, the deviation is reported to A. If her guess is correct, she is told so and the deviation is not reported to A.

Note that in order to make the simulation work using this idea the simulator must know c_i for all $i \in \mathcal{S}$ where $\text{col}(i) \neq \text{col}_0$ even if $i \notin \mathcal{M}$, i.e. it must know c_i for all $i \in \mathcal{M} \cup \mathcal{N}$. Thus to make the simulation work functionality $\mathcal{F}_{\text{IB2}}$ will leak the bits c_i $i \in \mathcal{N}$ "for free".

Honest-Parties

As in the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality.

Corrupted Parties

1. If A is corrupted: As in the leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality.
2. If B^* is corrupted:
   (a) The functionality waits to give output till B^* inputs $(\texttt{colors}, \text{col})$ where col is a coloring function.
   (b) The functionality samples a uniformly random pairing $\pi : [2\tau] \to [2\tau]$ and outputs $(\texttt{pairs}, \pi)$ to B^*. Let $\mathcal{S} = \mathcal{S}_\pi$, $\mathcal{M} = \mathcal{M}_{\pi,\text{col}}$ and $\mathcal{N} = \mathcal{N}_{\pi,\text{col}}$.
   (c) The functionality then waits for B^* to input $(\texttt{guess}, (g_i)_{i\in\mathcal{M}})$.
   (d) The functionality samples $(c_i)_{i\in[2\tau]} \in_{\text{R}} \{0,1\}^{2\tau}$. Then the functionality lets $c = 1$ if $g_i = c_i$ for all $i \in \mathcal{M}$, otherwise it lets $c = 0$. If $c = 0$ the functionality outputs $\texttt{fail}$ to A and terminates. Otherwise, the functionality determines col_0 and for $i \in \mathcal{N}$, the functionality outputs (i, c_i) to B^*.
   (e) The functionality waits for B^* to input $(\Gamma, (L_i)_{i\in[2\tau]})$ where $\Gamma, L_i \in \{0,1\}^\ell$.
   (f) For $i \in \mathcal{S}$ the functionality computes $N_{c_i}^i = L_i \oplus c_i\Gamma$ and outputs $(N_{c_i}^i, c_i)_{i\in\mathcal{S}}$ to A.

Figure 3.8: The Second Intermediate Functionality $\mathcal{F}_{\text{IB2}}$

Lemma 3.2. *$\mathcal{F}_{\text{IB2}}$ is linear locally reducible to $\mathcal{F}_{\text{IB1}}$.*

Proof. To implement $\mathcal{F}_{\text{IB2}}$ simply call $\mathcal{F}_{\text{IB1}}$. Note that the simulator must simulate $\mathcal{F}_{\text{IB2}}$ to the environment and fully controls the $\mathcal{F}_{\text{IB1}}$ towards the corrupted B^*.

First the simulator observes the input $(\texttt{colors}, \text{col}, (\Lambda_i, L_i)_{i\in 2\tau})$ of B^* to $\mathcal{F}_{\text{IB1}}$ and inputs $(\texttt{colors}, \text{col})$ to $\mathcal{F}_{\text{IB2}}$. $\mathcal{F}_{\text{IB2}}$ outputs $(\texttt{pairs}, \pi)$ and the simulator inputs $(\texttt{pairs}, \pi)$ to B^* on behalf of $\mathcal{F}_{\text{IB1}}$, and computes $\mathcal{M}$ as $\mathcal{F}_{\text{IB1}}$ and $\mathcal{F}_{\text{IB2}}$ would have done. Then the simulator observes the input $(\texttt{guess}, (g_i)_{i\in\mathcal{M}})$ from B^* to $\mathcal{F}_{\text{IB1}}$ and inputs $(\texttt{guess}, (g_i)_{i\in\mathcal{M}})$ to $\mathcal{F}_{\text{IB2}}$. If $\mathcal{F}_{\text{IB2}}$ outputs `fail` to A the simulation is over, and it is perfect as $\mathcal{F}_{\text{IB1}}$ and $\mathcal{F}_{\text{IB2}}$ fail based on the same event. If $\mathcal{F}_{\text{IB2}}$ does not fail it determines col_0 and for $i \in \mathcal{M}$, if $\text{col}(i) \neq \text{col}_0$, the functionality outputs (i, c_i) to the simulator. Note that the simulator can also determine col_0 from col.

Now let $\Gamma = \Lambda_{\text{col}_0}$ and for $i \in \mathcal{M} \cup \mathcal{N}$, if $\text{col}(i) = \text{col}_0$, let $L'_i = L_i$. Then for $i \in \mathcal{M} \cup \mathcal{N}$, if $\text{col}(i) \neq \text{col}_0$, let $L'_i = (L_i \oplus c_i\Lambda_{\text{col}(i)}) \oplus c_i\Gamma$. Finally input $(\Gamma, (L'_i)_{i\in[2\tau]})$ to $\mathcal{F}_{\text{IB2}}$.

As a result $\mathcal{F}_{\text{IB2}}$ will for $i \in \mathcal{S}$ where $\text{col}(i) = \text{col}_0$, output $L'_i \oplus c_i\Gamma = L_i \oplus c_i\Lambda_{\text{col}_0}$, and for for $i \in \mathcal{S}$ where $\text{col}(i) \neq \text{col}_0$ it will output $L'_i \oplus c_i\Gamma = L_i \oplus c_i\Lambda_{\text{col}(i)}$. Hence $\mathcal{F}_{\text{IB2}}$ gives exactly the outputs that $\mathcal{F}_{\text{IB1}}$ would have given after interacting with B^*, giving a perfect simulation. □

The Leakage Agent

Finally we now specify a leakage agent LA so that the LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ functionality is equivalent to $\mathcal{F}_{\text{IB2}}$.

The interaction between LA and the adversary is as follows.

1. LA waits for the adversary to input $(\texttt{colors}, \text{col})$ where col is a function $[2\tau] \to [2\tau]$.
2. Then LA outputs $(\texttt{pairs}, \pi)$ to the adversary, where $\pi : [2\tau] \to [2\tau]$ is a uniformly random pairing as defined above.
3. LA waits for the adversary to input $(\texttt{guess}, (g_i)_{i\in\mathcal{M}_{\pi,\text{col}}})$.

After this interaction and on input $(c_i)_{i\in[\tau]} \in \{0,1\}^\tau$ the leakage agent LA does the following. Here we let $\Pi : \mathcal{S} \to [\tau]$ be an order preserving function (simply to map $\mathcal{S}$ onto $[\tau]$).

1. LA computes $\mathcal{M} = \mathcal{M}_{\pi,\text{col}}$ and $\mathcal{N} = \mathcal{N}_{\pi,\text{col}}$. If $\bigwedge_{i\in\mathcal{M}}(g_{\Pi(i)} = c_i)$ sets $c = 1$, otherwise it sets $c = 0$.
2. LA computes $S = \{j = \Pi(i) | i \in \mathcal{M} \cup \mathcal{N}\}$ and outputs (c, S).

Notice that when we define LA in this way $\mathcal{F}_{\text{IB2}}$ and LA-leaky $\mathcal{F}_{\Gamma\text{-ROT}}$ is exactly the same. Therefore we trivially get the following.

Lemma 3.3. *For* LA *as defined above the* LA*-leaky* $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ *functionality is linear locally equivalent to the* $\mathcal{F}_{\text{IB2}}$ *functionality.*

And this leads us to the following theorem.

Theorem 3.4. *For* LA *as defined above* LA*-leaky* $\mathcal{F}_{\Gamma\text{-ROT}}(\tau, \ell)$ *is linear reducible to* $(\mathcal{F}_{\text{OT}}(2\tau, \ell), \mathcal{F}_{\text{EQ}}(\tau\ell))$.

Proof. This follows from Lemma 3.1, 3.2 and 3.3 and by the transitivity of linear local reducibility.

□

What remains then to complete the proof of Theorem 3.3 is to prove that LA defined in this way is $\kappa = \frac{3}{4}\tau$-secure.

Lemma 3.4. *Let* B^* *be an adversary playing in* LeakageGame$(\mathsf{LA}, \mathsf{B}^*, \tau)$. *Assume that during the interaction between* B^* *and* LA, B^* *inputs* $(\texttt{colors}, \text{col})$ *and* LA *outputs* $(\texttt{pairs}, \pi)$. *Let* $\mathcal{S} = \mathcal{S}_\pi$, $\mathcal{M} = \mathcal{M}_{\text{col},\pi}$ *and* $\mathcal{N} = \mathcal{N}_{\text{col},\pi}$. *Then the success probability of* B^* *is at most* $2^{-|\mathcal{S}\setminus\mathcal{N}|}$.

Proof. Let $\mathcal{S}' = \Pi(\mathcal{S})$, $\mathcal{M}' = \Pi(\mathcal{M})$ and $\mathcal{N}' = \Pi(\mathcal{N})$, i.e. $\mathcal{S}', \mathcal{M}'$ and $\mathcal{N}'$ are essentially the same sets as $\mathcal{S}, \mathcal{M}$ and $\mathcal{N}$ only mapped to interval $[\tau]$ instead of $[2\tau]$. In order for B^* to win the game LA must set $c = 1$. For this to happen B^* must guess the uniformly random bits $(c_i)_{i\in\mathcal{M}'}$. This happens with probability $2^{-|\mathcal{M}|}$. Furthermore, to win B^* must also guess the bits $(c_i)_{i\in\mathcal{S}'\setminus(\mathcal{M}'\cup\mathcal{N}')}$. This happens with probability $2^{-|\mathcal{S}\setminus(\mathcal{M}\cup\mathcal{N})|}$. Since the sets $\mathcal{M}$ and $\mathcal{N}$ are disjoint, this gives a total success probability of $2^{-|\mathcal{S}\setminus\mathcal{N}|}$.

□

Now let π be chosen uniformly at random. It is then easy to see that the probability of B^* winning LeakageGame$(\mathsf{LA}, \mathsf{B}^*, \tau)$ when she picks a coloring col is at most

$$\text{success}_{\text{col}} = \sum_{p=0}^{\tau} \Pr\left(|\mathcal{S}\setminus\mathcal{N}| = p\right) 2^{-p} \ .$$

For each size p, let P_p be an index variable which is 1 if $|\mathcal{S}\setminus\mathcal{N}| = p$ and 0 otherwise. Note that $\mathrm{E}\left[P_p\right] = \Pr\left(|\mathcal{S}\setminus\mathcal{N}| = p\right)$, and note that $\sum_{p=0}^{\tau} P_p 2^{-p} = 2^{-|\mathcal{S}\setminus\mathcal{N}|}$ as $P_p = 0$ for $p \neq |\mathcal{S}\setminus\mathcal{N}|$ and $P_p = 1$ for $p = |\mathcal{S}\setminus\mathcal{N}|$. Then

$$\begin{aligned}
\text{success}_{\text{col}} &= \sum_{p=0}^{\tau} \Pr\left(|\mathcal{S}\setminus\mathcal{N}| = p\right) 2^{-p} \\
&= \sum_{p=0}^{\tau} \mathrm{E}\left[P_p\right] 2^{-p} \\
&= \mathrm{E}\left[\sum_{p=0}^{\tau} P_p 2^{-p}\right] \\
&= \mathrm{E}\left[2^{-|\mathcal{S}\setminus\mathcal{N}|}\right] \ .
\end{aligned}$$

Now since $\phi(x) = 2^{-x}$ is a concave function we have by Jensen's inequality that

$$\text{success}_{\text{col}} = \mathrm{E}\left[2^{-|\mathcal{S}\setminus\mathcal{N}|}\right] \leq 2^{-\mathrm{E}[|\mathcal{S}\setminus\mathcal{N}|]} \ .$$

It follows that if we can compute $m_0 = \min_{\text{col}}(\mathrm{E}\left[|\mathcal{S}_\pi \setminus \mathcal{N}_{\pi,col}|\right])$, then 2^{-m_0} is an upper bound on the success probability of any adversary B^*. Thus if we can show that $m_0 \geq \frac{3}{4}\tau$ we have that LA is $\kappa = \frac{3}{4}\tau$-secure. We give this bound in Lemma 3.5.

Lemma 3.5. *For uniformly random pairing π we have*

$$\mathrm{E}\left[|\mathcal{S}_\pi \setminus \mathcal{N}_{\pi,\text{col}}|\right] \geq \frac{3}{4}\tau = \kappa \ ,$$

for any coloring function col*.*

Proof. Let col_i for $i \in [2\tau - 1]$ denote each of the colors different from col_0. Furthermore, let C_i be the balls in $[2\tau]$ of color col_i, i.e., $C_i = \{j \in [2\tau] | \, \text{col}(j) = \text{col}_i\}$. Also consider each ball j such that j or $\pi(j)$ is in $\mathcal{S} \setminus \mathcal{N}$. Denote this set B, i.e.,

$$B = |\{j \in [2\tau] | \{j, \pi(j)\} \cap (\mathcal{S} \setminus \mathcal{N}) \neq \emptyset\} \ .$$

Note that by definition of $\mathcal{S}$ only one ball, j or $\pi(j)$, can be in $\mathcal{S} \setminus \mathcal{N}$. Therefore, if we count the balls in B we find that $|B| = 2|(\mathcal{S} \setminus \mathcal{N})|$. I.e. in counting the size of B we count each member of $\mathcal{S}$ twice.

Now note that the set of all balls $[2\tau]$ is a disjoint union of $C_0, \ldots, C_{2\tau-1}$. This means that we can write

$$|\mathcal{S} \setminus \mathcal{N}| = \frac{1}{2}|B| = \frac{1}{2}\sum_{i=0}^{2\tau-1} |C_i \cap B| \ .$$

Hence we have that

$$\mathrm{E}\left[|\mathcal{S} \setminus \mathcal{N}|\right] = \frac{1}{2}\sum_{i=0}^{2\tau-1} \mathrm{E}\left[|C_i \cap B|\right] \ .$$

We will use this to get a bound on $\mathrm{E}\left[|\mathcal{S} \setminus \mathcal{N}|\right]$.

By definition of $\mathcal{N}$ and $\mathcal{S}$ the set $\mathcal{S} \setminus \mathcal{N}$ is the set of pairs that are either mismatched or of color col_0. This means that

$$|C_0 \cap B| = |C_0| \ .$$

Furthermore, it means that for all $i \in [2\tau - 1]$ and each $j \in C_i$ we have that $j \in C_i \cap B$ iff $\text{col}(j) \neq \text{col}(\pi(j))$. Since $\pi(j)$ is uniform in $[2\tau] \setminus \{j\}$ and there are a total of $|C_i| - 1$ balls of color col_i in $[2\tau] \setminus \{j\}$ we have that

$$\Pr\left(j \in C_i \cap B\right) = \frac{(2\tau - 1) - (|C_i| - 1)}{2\tau - 1} = \frac{2\tau - |C_i|}{2\tau - 1} \ .$$

This implies that

$$\mathrm{E}\left[|C_i \cap B|\right] = |C_i|\frac{2\tau - |C_i|}{2\tau - 1} = (2\tau|C_i| - |C_i|^2)\frac{1}{2\tau - 1} \ ,$$

and hence

$$\begin{aligned}\sum_{i=1}^{2\tau-1} \mathrm{E}\left[|C_i \cap B|\right] &= \frac{1}{2\tau-1}\sum_{i=1}^{2\tau-1}(2\tau|C_i| - |C_i|^2) \\ &= \frac{1}{2\tau-1}(2\tau\sum_{i=1}^{2\tau-1}|C_i| - \sum_{i=1}^{2\tau-1}|C_i|^2) \\ &= \frac{1}{2\tau-1}(2\tau(2\tau-|C_0|) - \sum_{i=1}^{2\tau-1}|C_i|^2)\ ,\end{aligned}$$

where the last equation follows from $\sum_{i=0}^{2\tau-1}|C_i| = 2\tau$. Combining the above equations we have that

$$\begin{aligned}\sum_{i=0}^{2\tau-1} \mathrm{E}\left[|C_i \cap B|\right] &= |C_0| + \frac{1}{2\tau-1}(2\tau(2\tau-|C_0|) - \sum_{i=1}^{2\tau-1}|C_i|^2) \\ &= |C_0| - \frac{2\tau}{2\tau-1}|C_0| + \frac{1}{2\tau-1}(4\tau^2 - \sum_{i=1}^{2\tau-1}|C_i|) \\ &= -\frac{1}{2\tau-1}|C_0| + \frac{1}{2\tau-1}(4\tau^2 - \sum_{i=1}^{2\tau-1}|C_i|^2) \\ &= \frac{4\tau^2}{2\tau-1} - \frac{1}{2\tau-1}(|C_0| + \sum_{i=1}^{2\tau-1}|C_i|^2)\ .\end{aligned}$$

To minimize this expression we have to maximize $|C_0| + \sum_{i=1}^{2\tau-1}|C_i|^2$. Recall that col_0 is defined to be the most common color, so we must have $|C_0| \geq |C_i|$ for $i > 0$. Under this restriction it is easy to see that $|C_0| + \sum_{i=1}^{2\tau-1}|C_i|^2$ is maximal when $|C_0| = |C_1| = \tau$ and $|C_2| = \cdots |C_{2\tau}| = 0$, in which case $|C_0| + \sum_{i=1}^{2\tau-1}|C_i|^2 = \tau + \tau^2$. So,

$$\begin{aligned}\mathrm{E}\left[|\mathcal{S} \setminus \mathcal{N}|\right] &= \frac{1}{2}\sum_{i=0}^{2\tau-1}\mathrm{E}\left[|C_i \cap B|\right] \\ &= \frac{1}{2}\left(\frac{4\tau^2}{2\tau-1} - \frac{1}{2\tau-1}(\tau+\tau^2)\right) \\ &= \frac{1}{2}\left(\frac{3\tau^2-\tau}{2\tau-1}\right) \\ &= \frac{1}{2}\tau\frac{3\tau-1}{2\tau-1} > \frac{1}{2}\tau\frac{3\tau}{2\tau} = \frac{3}{4}\tau = \kappa\end{aligned}$$

□

3.6 $\mathcal{F}_{\Delta\text{-ROT}}$ From Leaky $\mathcal{F}_{\Delta\text{-ROT}}$

Finally we will give a functionality $\mathcal{F}_{\Delta\text{-ROT}}$ which does not leak any bits of the global key. I.e. the LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality where LA is the leakage

agent that leaks nothing. This functionality will be useful in Chapter 4 when we prove security of the TinyOT protocol. We present the non-leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality in Figure 3.9.

Honest Parties

On input `start` from both A and B the functionality does the following.

1. The functionality samples $\Delta \in_R \{0,1\}^\tau$ and outputs it to A.
2. For all $i \in [\ell]$ the functionality samples $b_i \in_R \{0,1\}$ and $M_0^i \in_R \{0,1\}^\tau$.
3. The functionality outputs $(M_0^i \oplus b_i\Delta, b_i)_{i\in[\ell]}$ to B and $(M_0^i)_{i\in[\ell]}$ to A.

Corrupted Parties

1. If B is corrupted, the functionality waits to give output till it receives the message $(\hat{M}^i_{\hat{b}_i}, \hat{b}_i)_{i\in[\ell]}$ from B, where $\hat{M}^i_{\hat{b}_i} \in \{0,1\}^\tau$ and $\hat{b}_i \in \{0,1\}$. The functionality then sets $b_i = \hat{b}_i$ and $M_0^i = \hat{M}^i_{b_i} \oplus b_i\Delta$ and outputs as described above.
2. If A is corrupted, the functionality waits to give output till it receives the message $(\hat{\Delta}, (\hat{M}_0^i)_{i\in[\ell]})$ from A, where $\hat{\Delta}, \hat{M}_0^i \in \{0,1\}^\tau$. The functionality then sets $\Delta = \hat{\Delta}$ and $M_0^i = \hat{M}_0^i$ and outputs as described above.

Figure 3.9: The $\mathcal{F}_{\Delta\text{-ROT}}(\ell,\tau)$ functionality

In Figure 3.10 we describe a protocol which takes a leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality, where one quarter of the bits of the global key might leak, and amplifies it to the non-leaky $\mathcal{F}_{\Delta\text{-ROT}}$ functionality. We prove the following theorem.

1. The parties invoke a LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell,\tau)$ with $\tau = \frac{22}{3}\psi$. The output to A is $(\hat{M}^i_{b_i}, b_i)_{i\in[\ell]}$. The output to B is $(\hat{\Delta}, (\hat{M}^i_0)_{i\in[\ell]})$.
2. B samples $\mathbf{A} \in_R \{0,1\}^{\psi\times\tau}$, a random binary matrix with ψ rows and τ columns, and sends $\mathbf{A}$ to A.
3. A computes $M^i_{b_i} = \mathbf{A}\hat{M}^i_{b_i} \in \{0,1\}^\psi$ and outputs $(M^i_{b_i}, b_i)_{i\in[\ell]}$.
4. B computes $\Delta = \mathbf{A}\hat{\Delta}$ and $M_0^i = \mathbf{A}\hat{M}_0^i$ and outputs $(\Delta, (\hat{M}_0^i)_{i\in[\ell]})$.

Figure 3.10: Protocol for reducing $\mathcal{F}_{\Delta\text{-ROT}}(\ell,\psi)$ to LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell,\tau)$.

Theorem 3.5. *Let $\tau = \frac{22}{3}\psi$ and LA be a $\left(\frac{3}{4}\tau\right)$-secure leakage agent on τ bits. The protocol in Figure 3.10 securely implements $\mathcal{F}_{\Delta\text{-ROT}}(\ell,\psi)$ in the LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell,\tau)$-hybrid model with security parameter ψ. The communication is $O(\psi^2)$ and the work is $O(\psi^2\ell)$.*

Correctness of the protocol is straight forward: We have that

$$\hat{M}^i_{b_i} = \hat{M}^i_0 \oplus b_i\hat{\Delta}\ ,$$

so

$$M^i_{b_i} = \mathbf{A}\hat{M}^i_{b_i} = \mathbf{A}M^i_0 \oplus b_i\mathbf{A}\hat{\Delta} = M^i_0 \oplus b_i\Delta.$$

In addition it is clear that the protocol leaks no information on the b_i's to B: there is only communication from B to A. It is therefore sufficient to look at the case where A is corrupted. To prove security against corrupted A^* we will

prove that Δ is uniformly random in the view of A^* except with probability $2^{2-\psi}$.

When we say that Δ *is uniform to* A^* we mean that Δ is uniformly random in $\{0,1\}^\psi$ and independent of the view of A^*. When we say *except with probability* $2^{2-\psi}$ we mean that there exists a failure event F for which it holds that:

1. F occurs with probability at most $2^{2-\psi}$.
2. When F does not occur, then Δ is uniform to A^*.

To prove this we consider the following experiment LeakExp. Assuming that LA is a leakage agent on τ bits which is κ-secure LeakExp corresponds to the leakage on $\hat{\Delta}$ a corrupt A^* receives during the protocol.

$$\begin{array}{l}\underline{\text{LeakExp}(\mathsf{LA},\mathsf{A}^*)}\\ \Delta \in_{\mathrm{R}} \{0,1\}^\tau\\ \mathsf{A}^*(\tau) \leftrightarrow \mathsf{LA}(\tau)\\ (S,c) \leftarrow \mathsf{LA}(\Delta)\\ \mathbf{A} \in_{\mathrm{R}} \{0,1\}^{\psi\times\tau}\\ \text{Input } \mathbf{A} \text{ to } \mathsf{A}^*\\ \text{Output } \Delta = \mathbf{A}\hat{\Delta}\end{array}$$

To show that Δ is uniform to A^*, we give the following three technical lemmas on on LeakExp.

For a subset $S \subset [\tau]$ of the column indices, let $\mathbf{A}^S$ be the matrix where column j is equal to the j'th column of $\mathbf{A}$ if $j \in S$, and column j is the 0 vector if $j \notin S$. We say that we *blind out* column j with 0's if $j \notin S$. Similarly, for a column vector v we use the notation v^S to mean that we set all indices v_i where $i \notin S$ to be 0. Note that $\mathbf{A}v^S = \mathbf{A}^S v = \mathbf{A}^S v^S$.

Lemma 3.6. *Let S and $\boldsymbol{A}$ be the values sampled in* LeakExp$(\mathsf{LA},\mathsf{A}^*)$*. If $\boldsymbol{A}^{\bar{S}}$ spans $\{0,1\}^\psi$, then Δ is uniform to A^*.*

Proof. We start by making two simple observations. First of all, if A^* learns $\hat{\Delta}_j$ for $j \in S$, then it learns $\hat{\Delta}^S$[2], so it knows $\mathbf{A}\hat{\Delta}^S = \mathbf{A}^S\hat{\Delta}$. The second observation is that $\mathbf{A}\hat{\Delta} = \mathbf{A}^S\hat{\Delta} + \mathbf{A}^{\bar{S}}\hat{\Delta}$, as $\mathbf{A} = \mathbf{A}^S + \mathbf{A}^{\bar{S}}$. The lemma follows directly from these observations and the premise: We have that $\mathbf{A}^{\bar{S}}\hat{\Delta}$ is uniformly random in $\{0,1\}^\psi$ when the columns of $\mathbf{A}^{\bar{S}}$ span $\{0,1\}^\psi$. Since $\mathbf{A}^{\bar{S}}\hat{\Delta} = \mathbf{A}\hat{\Delta}^{\bar{S}}$ and $\hat{\Delta}^{\bar{S}}$ is uniformly random and independent of the view of A^* it follows that $\mathbf{A}^{\bar{S}}\hat{\Delta}$ is uniformly random and independent of the view of A^*. Since only $\mathbf{A}^S\hat{\Delta}$ is known by A^* it follows that $\mathbf{A}^S\hat{\Delta} + \mathbf{A}^{\bar{S}}\hat{\Delta}$ is uniform to A^*. The proof concludes by using that $\Delta = \mathbf{A}^S\hat{\Delta} + \mathbf{A}^{\bar{S}}\hat{\Delta}$.

□

Lemma 3.7. *Let $n = \frac{9}{2}\psi$, $\alpha = \frac{44}{27}$ and $\kappa = \frac{3}{4}\tau$ s.t. $\tau = \alpha n$ and LA is κ-secure. Let W be the event that $|S| \geq \tau - n$ and $c = 1$ (where S and c are sampled as in* LeakExp$(\mathsf{LA},\mathsf{A}^*)$*). Then* $\Pr(W) \leq 2^{-\psi}$.

[2]Here we are looking at the string $\hat{\Delta}$ as a column vector of bits.

Proof. Without loss of generality we can assume that A^* interacts optimally with LA, i.e., $\log_2(\mathrm{E}\left[c2^{|S|}\right]) = \mathrm{leak}_{\mathsf{LA}}$. Since LA is κ secure on τ bits, it follows that $\mathrm{leak}_{\mathsf{LA}} \leq \tau - \kappa = \frac{1}{4}\tau$. This gives that

$$\mathrm{E}\left[c2^{|S|}\right] \leq 2^{\frac{1}{4}\tau} \ , \tag{3.2}$$

which we use later.

Now let $\bar{W}$ be the event that W does not happen. By the properties of conditional expected value we have that

$$\mathrm{E}\left[c2^{|S|}\right] = \Pr(W)\,\mathrm{E}\left[c2^{|S|}|W\right] + \Pr\left(\bar{W}\right)\mathrm{E}\left[c2^{|S|}|\bar{W}\right] \ .$$

When W happens, then $|S| \geq \tau - n = (\alpha - 1)n$ and $c = 1$, so $c2^{|S|} = 2^{|S|} \geq 2^{(\alpha-1)n}$. This gives that

$$\mathrm{E}\left[c2^{|S|}|W\right] \geq 2^{(\alpha-1)n} \ .$$

Hence

$$\mathrm{E}\left[c2^{|S|}\right] \geq \Pr(W)\,2^{(\alpha-1)n} \ .$$

Combining with (3.2) we get that

$$\Pr(W) \leq 2^{\frac{1}{4}\tau-(\alpha-1)n} \ .$$

It is, therefore, sufficient to show that $\frac{1}{4}\tau-(\alpha-1)n = -\psi$, which can be verified to be the case by definition of τ, α, n and ψ. □

Lemma 3.8. *As above let $n = \frac{9}{2}\psi$. Furthermore, let $x_1, \ldots, x_n \in_{\mathrm{R}} \{0,1\}^\psi$. Then $x_1, \ldots, x_n$ spans $\{0,1\}^\psi$ except with probability $2^{1-\psi}$.*

Proof. Define random variables $Y_1, \ldots, Y_n$ where $Y_i = 0$ if $x_1, \ldots, x_{i-1}$ spans $\{0,1\}^\psi$ or the span of $x_1, \ldots, x_{i-1}$ does not include x_i. Let $Y_i = 1$ in all other cases. Note that if $x_1, \ldots, x_{i-1}$ spans $\{0,1\}^\psi$, then $\Pr(Y_i = 1) = 0 \leq \frac{1}{2}$ and that if $x_1, \ldots, x_{i-1}$ does not span $\{0,1\}^\psi$, then they span at most half of the vectors in $\{0,1\}^\psi$ and hence again $\Pr(Y_i = 1) \leq \frac{1}{2}$. This means that it holds for all Y_i that $\Pr(Y_i = 1) \leq \frac{1}{2}$ independently of the values of Y_j for $j \neq i$. This implies that if we let $Y = \sum_{i=1}^n Y_i$, then

$$\Pr\left(Y \geq \frac{1}{2}(a+n)\right) \leq 2e^{-a^2/2n} \ ,$$

using the random walk bound. Namely, let $X_i = 2Y_i - 1$. Then $X_i \in \{-1, 1\}$ and it holds for all i that $\Pr(X_i = 1) \leq \frac{1}{2}$ independently of the other X_j. If the X_i had been independent and $\Pr(X_i = 1) = \Pr(X_i = -1) = \frac{1}{2}$, and $X = \sum_{i=1}^n X_i$, then the random walk bound gives that

$$\Pr(X \geq a) \leq 2e^{-a^2/2n} \ .$$

Since we have that $\Pr(X_i = 1) \leq \frac{1}{2}$ independently of the other X_j, the upper bound applies also to our setting. Then use that $X = 2Y - n$.

If we let $a = \frac{5}{2}\psi$, then

$$\frac{1}{2}(a+n) = \frac{7}{2}\psi = n - \psi$$

and

$$2e^{-a^2/2n} = 2e^{-\left(\frac{5}{2}\psi\right)^2/2\frac{9}{2}\psi} = 2e^{-\frac{25}{36}\psi} ,$$

and $e^{-\frac{25}{36}} < \frac{1}{2}$. It follows that $\Pr[Y \geq n - \psi] \leq 2^{1-\psi}$. When $Y \leq n - \psi$, then $Y_i = 0$ for at least ψ values of i. This is easily seen to imply that $x_1, \ldots, x_n$ contains at least ψ linear independent vectors.

□

Given the lemmas above we are now ready to conclude the proof of Theorem 3.5 for corrupt A^*.

Proof. As mentioned above we will not give a simulation argument but just prove that Δ is uniformly random in the view of a corrupt A^* except with probability $2^{2-\psi}$. Turning this argument into a simulation argument is straight forward.

Recall that W is the event that $|S| \geq \tau - n$ and $c = 1$. By Lemma 3.7 we have that $\Pr[W] \leq 2^{-n} \leq 2^{-\psi}$. For the rest of the analysis we assume that W does not happen, i.e., $|S| < \tau - n$ and hence $|\bar{S}| \geq \tau = \frac{9}{2}\psi$. Since $\mathbf{A}$ is picked uniformly at random and independent of S it follows that $\frac{9}{2}\psi$ of the columns in $\mathbf{A}^{\bar{S}}$ are uniformly random and independent. Hence, by Lemma 3.8, they span $\{0,1\}^\psi$ except with probability $2^{1-\psi}$. We let D be the event that they do not span. If we assume that D does not happen, then by Lemma 3.6 Δ is uniform to A^*. I.e., if the event $F = W \cup D$ does not happen, then Δ is uniform to A^*. Since

$$\Pr[F] \leq \Pr[W] + \Pr[D] \leq 2^{-\psi} + 2^{1-\psi} \leq 2^{2-\psi} ,$$

we have that Δ is uniform to A^* with overwhelming probability. □

Similar to Cor. 3.1 we can derive the following corollary for the $\mathcal{F}_{\Delta\text{-ROT}}$ functionality.

Corollary 3.2. *Let ψ denote the security parameter and let $\ell = \text{poly}(\psi)$. The functionality $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \psi)$ can be reduced to $(\mathcal{F}_{\text{OT}}(2\tau, \psi), \mathcal{F}_{\text{EQ}}(\psi))$. The communication is $O(\psi\ell + \psi^2)$ and the work is $O(\psi^2\ell)$.*

Proof. Combining Theorem 3.5, 3.2 and 3.3 we have that $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \psi)$ can be reduced to $(\mathcal{F}_{\text{OT}}(\frac{44}{3}\psi, \ell), \mathcal{F}_{\text{EQ}}(\frac{22}{3}\psi\ell))$ with communication $O(\psi^2)$ and work $O(\psi^2\ell)$. For any polynomial ℓ, we can implement $\mathcal{F}_{\text{OT}}(\frac{44}{3}\psi, \ell)$ given $\mathcal{F}_{\text{OT}}(\frac{44}{3}\psi, \psi)$ and a pseudo-random generator $\text{prg} : \{0,1\}^\psi \to \{0,1\}^\ell$. Namely, seeds are sent using the OTs and the prg is used to one-time pad encrypt the messages. The communication is 2ℓ. If we use the RO to implement the pseudo-random generator and count the hashing of ψ bits as $O(\psi)$ work, then the work is $O(\ell\psi)$ (using ℓ/ψ calls to H to expand ψ bit seeds to ℓ bits). We can implement $\mathcal{F}_{\text{EQ}}(\frac{22}{3}\psi\ell)$ by comparing short hashes produced using the RO. The work is $O(\psi\ell)$ (using $\frac{22}{3}\ell$ calls to H to hash the $\frac{22}{3}\psi\ell$-bit string down to ψ-bits).

□

3.7 Complexity Analysis

We sketch a complexity analysis of the protocol in terms of the number of calls to the hash function H. We will count the number of total calls made by both A and B:

To implement $\mathcal{F}_{\text{ROT}}(\ell, \psi)$ from LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \tau)$ for ψ-secure LA, in Section 3.2 we used 3ℓ calls to H. From Cor. 3.1 we have that LA-leaky $\mathcal{F}_{\Delta\text{-ROT}}(\ell, \frac{4}{3}\psi)$ for ψ-secure LA, can be reduced to $(\mathcal{F}_{\text{OT}}(\frac{8}{3}\psi, \psi), \mathcal{F}_{\text{EQ}}(\psi))$. Doing this involves using H to expand 3 ψ-bits seeds to ℓ bits for each OT, meaning a total of 8ℓ calls to H. We also need to use H to hash a $\frac{4}{3}\psi\ell$-bit string to ψ-bits. Both A and B must do this giving a total of $\frac{8}{3}\ell$ calls to H. Thus in total we get $(3 + \frac{8}{3} + 8)\ell \leq 14\ell$ to calls H or 14 calls to H pr. ROT, apart from the cost of $(\mathcal{F}_{\text{OT}}(\frac{8}{3}\psi, \psi), \mathcal{F}_{\text{EQ}}(\psi))$.

To implement non-leaky version of $\mathcal{F}_{\Delta\text{-ROT}}$ as in Section 3.6 we will need somewhat longer keys in the leaky-$\mathcal{F}_{\Delta\text{-ROT}}$ functionality, meaning more work to expand seeds and so on. Doing a similar calculation to the one above, we get that the price for this functionality is around 59ℓ calls to H or 59 calls to H pr. Δ-ROT.

Chapter 4

TinyOT

In this chapter we present the TinyOT protocol. As described in Chapter 1, the protocol is essentially the classic semi-honestly secure protocol of [GMW87], where all shared bits are authenticated using MACs in order to gain malicious security. Using the MACs we make sure that a malicious adversary can never deviate from the protocol with out being detected, as this would correspond to forging a MAC.

The MACs are of the following form: If A holds a bit b then the MAC on b is a string $M_b = K_b \oplus b\Delta \in \{0,1\}^\psi$ where K_b and Δ_A are uniformly random strings held by B. Here K_b is a *local key* for b and Δ_A is a *global key* used for all of A's MACs. Intuitively, forging such a MAC should be equivalent to A guessing the value of Δ_A. The MACs are defined symmetrically for B.

Apart from a way to authenticate bits in this way, we will need two additional tools to implement the TinyOT protocol: namely a way to do authenticated local ands (aANDs) and authenticated OTs (aOTs). I.e. for aANDs we will need a protocol so that if A has authenticated bits a and b (i.e. she has valid MACs on these bits) she can obtain the authenticated bit $c = ab$. For aOT we need protocol, so that if A as authenticated message bits m_0 and m_1 and B has authenticated choice bit b, B can obtain an authenticated bit $z = m_b$.

Given just these tools we have the TinyOT protocol.

Overview

We present the TinyOT protocol in a top-down fashion as follows.

- In Section 4.1 we first show how to implement a malicious secure 2PC protocol using a *dealer* functionality that provides random bit authentications (aBits), authenticated local ANDs (aANDs) and authenticated OTs (aOTs). The argument for security is essentially as stated above: a malicious adversary can never deviate from the protocol without being detected, as this would correspond to forging a MAC. As forging a MAC involves guessing the random string $\Delta \in \{0,1\}^\psi$ this can only happen with negligible probability.

- Then in Section 4.2 we start implementing the dealer functionality by

giving a way to produce random aBits. This turns out to be equivalent Δ-ROT functionality we implemented very efficiently in Chapter 3.

- In Section 4.3 we extend the dealer functionality with a way of giving random aANDs. We do this in a two step fashion similar to the LEGO-style cut-n-choose method: we first produce a large amount aANDs in a slightly naive way such that up to σ them may be insecure. Then we combine these into a slightly smaller amount of fully maliciously secure aANDs.

- In Section 4.4 we finish the implementation of the the dealer functionality by extending it with a way to give random aOTs. As for aANDs we use an approach similar to the LEGO-style cut-n-choose.

- In Section 4.5 we sketch a complexity analysis of the protocol counting the symmetric primitives used in the protocol.

- Finally in Section 4.6 we report on our experimental implementation of the TinyOT protocol. We briefly discuss the implementation and give benchmarks for oblivious AES evaluation for different parameters.

4.1 Secure Two-Party in the Dealer Model

We want to implement the functionality $\mathcal{F}_{\text{2PC}}$ for Boolean two-party secure computation as described in Figure 4.2. We will implement the $\mathcal{F}_{\text{2PC}}$ functionality in the $\mathcal{F}_{\text{DEAL}}$-hybrid model of Figure 4.3. The $\mathcal{F}_{\text{DEAL}}$ functionality provides the parties with aBits, aANDs and aOTs, and models the preprocessing phase of our protocol. We introduce notation in Figure 4.1 for working with authenticated bits. The protocol implementing $\mathcal{F}_{\text{2PC}}$ in the dealer model is described in Figure 4.4. The dealer offers random authenticated bits (to A or B), random authenticated local AND triples and random authenticated OTs. Those are all the ingredients that we need to build the 2PC protocol. Note that the dealer offers randomized versions of all commands: this is not a problem as the "standard" version of the commands (the one where the parties can specify their input bits instead of getting them at random from the functionality) are linearly reducible to the randomized version, as can be easily deduced from the protocol description. We prove the following theorem.

Theorem 4.1. *The protocol in Figure 4.4 securely implements the functionality $\mathcal{F}_{\text{2PC}}$ in the $\mathcal{F}_{\text{DEAL}}$-hybrid model with security parameter ψ.*

Proof. The simulator can be built in a standard way, incorporating the $\mathcal{F}_{\text{DEAL}}$ functionality and learning all the shares, keys and MACs that the adversary was supposed to use in the protocol.

In a little more detail, knowing all outputs from $\mathcal{F}_{\text{DEAL}}$ to the corrupted parties allows the simulator to extract inputs used by corrupted parties and input these to the functionality $\mathcal{F}_{\text{2PC}}$ on behalf of the corrupted parties. As an example, if A is corrupted, then learn the x_A sent to A by $\mathcal{F}_{\text{DEAL}}$ in **Input** and

Global Key

We call $\Delta_A, \Delta_B \in \{0,1\}^\psi$ the two *global keys*, held by B and A respectively.

Authenticated Bit

- We write $[x]_\mathsf{A}$ to represent an *authenticated secret bit* held by A. Here B knows a key $K_x \in \{0,1\}^\psi$ and A knows a bit x and a MAC $M_x = K_x \oplus x\Delta_A \in \{0,1\}^\psi$. Let $[x]_\mathsf{A} \stackrel{\text{def}}{=} (x, M_x, K_x)$.[a]
- If $[x]_\mathsf{A} = (x, M_x, K_x)$ and $[y]_\mathsf{A} = (y, M_y, K_y)$ we write $[z]_\mathsf{A} = [x]_\mathsf{A} \oplus [y]_\mathsf{A}$ to indicate $[z]_\mathsf{A} = (z, M_z, K_z) \stackrel{\text{def}}{=} (x \oplus y, M_x \oplus M_y, K_x \oplus K_y)$. Note that no communication is required to compute $[z]_\mathsf{A}$ from $[x]_\mathsf{A}$ and $[y]_\mathsf{A}$.
- We authenticate a constant bit (a value known both to A and B) $b \in \{0,1\}$ as follows: A sets $M_b = 0^\psi$, B sets $K_b = b\Delta_A$, now $[b]_\mathsf{A} \stackrel{\text{def}}{=} (b, M_b, K_b)$. For a constant b we let $[x]_\mathsf{A} \oplus b \stackrel{\text{def}}{=} [x]_\mathsf{A} \oplus [b]_\mathsf{A}$, and we let $b[x]_\mathsf{A}$ be equal to $[0]_\mathsf{A}$ if $b = 0$ and $[x]_\mathsf{A}$ if $b = 1$.
- We say that A *reveals* $[x]_\mathsf{A}$ by sending (x, M_x) to B who aborts if $M_x \neq K_x \oplus x\Delta_A$. Alternatively we say that A *announces* x by sending x to B without a MAC.
- Authenticated bits belonging to B are written as $[y]_\mathsf{B}$ and are defined symmetrically, changing side of all the values and using the global value Δ_B instead of Δ_A.

Authenticated Share

- We write $[x]$ to represent the situation where A and B hold $[x_A]_\mathsf{A}, [x_B]_\mathsf{B}$ and $x = x_A \oplus x_B$, and we write $[x] = ([x_A]_\mathsf{A}, [x_B]_\mathsf{B})$ or $[x] = [x_A|x_B]$.
- If $[x] = [x_A|x_B]$ and $[y] = [y_A|y_B]$ we write $[z] = [x] \oplus [y]$ to indicate $[z] = ([z_A]_\mathsf{A}, [z_B]_\mathsf{B}) = ([x_A]_\mathsf{A} \oplus [y_A]_\mathsf{A}, [x_B]_\mathsf{B} \oplus [y_B]_\mathsf{B})$. Note that no communication is required to compute $[z]$ from $[x]$ and $[y]$.
- We create an authenticated share of a constant $b \in \{0,1\}$ as follows: A and B create $[b] = [b|0]$. For a constant $b \in \{0,1\}$, we define $b[x]$ to be equal to $[0]$ if $b = 0$ and $[x]$ if $b = 1$.
- When an authenticated share is *revealed*, the parties reveal to each other their authenticated bits and abort if the MACs are not correct.

[a] Since Δ_A is a global value we will not always write it explicitly. Note that in $x\Delta_A$, x represents a *value*, 0 or 1, and that in $[x]_\mathsf{A}$, K_x and M_x it represents a *variable name*. I.e., there is only one key (MAC) per authenticated bit, and for the bit named x, the key (MAC) is named K_x (M_x). If $x = 0$, then $M_x = K_x$. If $x = 1$, then $M_x = K_x \oplus \Delta_A$.

Figure 4.1: Notation for authenticated and shared bits.

observe the value x_B sent by A to B. Then input $x = x_A \oplus x_B$ to $\mathcal{F}_{\text{2PC}}$. This is the same value as shared by $[x] = [x_A|x_B]$ in the protocol.

Honest parties are run on uniformly random inputs, and when a honest party (A say) is supposed to help open $[x]$, then the simulator learns from $\mathcal{F}_{\text{2PC}}$ the value x' that $[x]$ should be opened to. Then the simulator computes the share x_B that B holds, which is possible from the outputs of $\mathcal{F}_{\text{DEAL}}$ to B. Then the simulator learns the key K_{x_A} that B uses to authenticate x_A, which can also be computed from the outputs of $\mathcal{F}_{\text{DEAL}}$ to B. Then the simulator lets $x_A = x' \oplus x_B$ and and lets $M_{x_A} = K_{x_A} \oplus x_A K_{x_A}$ and sends (x_A, M_{x_A}) to B.

The simulator aborts if the adversary ever successfully sends some inconsistent bit, i.e., a bit different from the bit it should send according to the protocol

Rand

On input $(\texttt{rand}, vid)$ from A and B, with vid a fresh identifier, the functionality picks $r \in_{\mathrm{R}} \{0,1\}$ and stores (vid, r).

Input

On input $(\texttt{input}, \mathsf{P}, vid, x)$ from $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ and $(\texttt{input}, \mathsf{P}, vid, ?)$ from the other party, with vid a fresh identifier, the functionality stores (vid, x).

XOR

On command $(\texttt{XOR}, vid_1, vid_2, vid_3)$ from both parties (if vid_1, vid_2 are defined and vid_3 is fresh), the functionality retrieves (vid_1, x), (vid_2, y) and stores $(vid_3, x \oplus y)$.

AND

On command $(\texttt{AND}, vid_1, vid_2, vid_3)$ from both parties (if vid_1, vid_2 are defined and vid_3 is fresh), the functionality retrieves (vid_1, x), (vid_2, y) and stores $(vid_3, x \wedge y)$.

Output

On input $(\texttt{output}, \mathsf{P}, vid)$ from both parties, with $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ (and vid defined), the functionality retrieves (vid, x) and outputs it to P.

At each command the functionality leaks to the environment which command is being executed (keeping the value x in **Input** secret), and delivers messages only when the environment says so.

Figure 4.2: The functionality $\mathcal{F}_{\mathrm{2PC}}$ for Boolean Two-party Computation.

and its outputs from $\mathcal{F}_{\mathrm{DEAL}}$.

It is easy to see that the protocol is passively secure and that if the adversary never sends such an inconsistent bit, then he is perfectly following the protocol up to input substitution. So, to prove security it is enough to prove that, in the real world protocol, the adversary can only get away with using an inconsistent bit with negligible probability. Note that, to get away with using an inconsistent bit includes the adversary providing a correct MAC for the inconsistent bit. By Cor. 4.1 we have that using an inconsistent bit is equivalent to guessing the global key Δ of the opposing player. Since all inputs to the adversary are independent of Δ the adversary can guess Δ with at most negligible probability. □

We now show that an adversary getting away with using an inconsistent bit in the protocol in Figure 4.4 is equivalent to guessing the global key Δ of the opposing player.

To formalize this claim we consider the following game $\mathrm{Game}_{I,I}$ played by an attacker A:

Global key: A global key $\Delta \leftarrow \{0,1\}^{\psi}$ is sampled with some distribution and A might get side information on Δ.

MAC query I: If A outputs a query $(\texttt{mac}, b, l)$, where $b \in \{0,1\}$ and l is a label which A did not use before, sample a fresh local key $K \in_{\mathrm{R}} \{0,1\}^{\psi}$, give $M = K \oplus b\Delta$ to A and store (l, K, b).

Initialize

On input (`init`) from A and (`init`) from B, the functionality samples $\Delta_A, \Delta_B \in \{0,1\}^\psi$, stores them and outputs Δ_B to A and Δ_A to B. If A (resp. B) is corrupted, she gets to choose Δ_B (resp. Δ_A).

Authenticated Bit (A)

On input (`aBIT`, A) from A and B, the functionality samples a random $[x]_\mathsf{A} = (x, M_x, K_x)$ with $M_x = K_x \oplus x\Delta_A$ and outputs it (x, M_x to A and K_x to B). If B is corrupted he gets to choose K_x. If A is corrupted she gets to choose (x, M_x), and the functionality sets $K_x = M_x \oplus x\Delta_A$.

Authenticated Bit (B)

On input (`aBIT`, B) from A and B, the functionality samples a random $[x]_\mathsf{B} = (x, M_x, K_x)$ with $M_x = K_x \oplus x\Delta_B$ and outputs it (x, M_x to B and K_x to A). As in **Authenticated Bit (A)**, corrupted parties can choose their own randomness.

Authenticated local AND (A)

On input (`aAND`, A) from A and B, the functionality samples random $[x]_\mathsf{A}$,$[y]_\mathsf{A}$ and $[z]_\mathsf{A}$ with $z = xy$ and outputs them. As in **Authenticated Bit (A)**, corrupted parties can choose their own randomness.

Authenticated local AND (B)

Defined symmetrically.

Authenticated OT (A-B)

On input (`aOT`, A, B) from A and B, the functionality samples random $[x_0]_\mathsf{A}$,$[x_1]_\mathsf{A}$,$[c]_\mathsf{B}$ and $[z]_\mathsf{B}$ with $z = x_c = c(x_0 \oplus x_1) \oplus x_0$ and outputs them. As in **Authenticated Bit**, corrupted parties can choose their own randomness.

Authenticated OT (B-A)

Defined symmetrically.[a]

Global Key Queries

The adversary can at any point input (A, Δ) and be told whether $\Delta = \Delta_B$. And it can at any point input (B, Δ) and be told whether $\Delta = \Delta_A$.

[a] The dealer offers aOTs in both directions. Notice that the dealer could offer aOT only in one direction and the parties could then "turn" them: as regular OT, aOT is symmetric as well.

Figure 4.3: The functionality $\mathcal{F}_{\text{DEAL}}$ for dealing preprocessed values.

Break query I: If A outputs a query $(\texttt{break}, a_1, l_1, \ldots, a_p, l_p, M')$, where p is some positive integer and values $(l_1, K_1, b_1), \ldots, (l_p, K_p, b_p)$ are stored, then let $K = \oplus_{i=1}^p a_i K_i$ and $b = \oplus_{i=1}^p a_i b_i$. If $M' = K \oplus (1 \oplus b)\Delta$, then A wins the game. This query can be used only once.

We want to prove that if any A can win the game with probability q, then there exist an adversary B which does not use more resources than A and which guesses Δ with probability q without doing any MAC queries. Informally this argues that breaking the scheme is linear equivalent to guessing Δ without seeing any MAC values.

For this purpose, consider the following modified game $\text{Game}_{II,II}$ played by

Initialize

When activated the first time, A and B activate $\mathcal{F}_{\text{DEAL}}$ and receive Δ_B and Δ_A respectively.

Rand

A and B ask $\mathcal{F}_{\text{DEAL}}$ for random authenticated bits $[r_A]_\mathsf{A}, [r_B]_\mathsf{B}$ and stores $[r] = [r_A | r_B]$ under *vid*.

Input

If $\mathsf{P} = \mathsf{A}$, then A asks $\mathcal{F}_{\text{DEAL}}$ for an authenticated bit $[x_A]_\mathsf{A}$ and announces (i.e., no MAC is sent together with the bit) $x_B = x \oplus x_A$, and the parties build $[x_B]_\mathsf{B}$ and define $[x] = [x_A | x_B]$. The protocol is symmetric for B.

XOR

A and B retrieve $[x], [y]$ stored under vid_1, vid_2 and store $[z] = [x] \oplus [y]$ under vid_3. For brevity we drop explicit mentioning of variable identifiers below.

AND

A and B retrieve $[x], [y]$ and compute $[z] = [xy]$ as follows:

1. The parties ask $\mathcal{F}_{\text{DEAL}}$ for a random AND triplet $[u]_\mathsf{A}, [v]_\mathsf{A}, [w]_\mathsf{A}$ with $w = uv$.
   A reveals $[f]_\mathsf{A} = [u]_\mathsf{A} \oplus [x_A]_\mathsf{A}$ and $[g]_\mathsf{A} = [v]_\mathsf{A} \oplus [y_A]_\mathsf{A}$.
   The parties compute $[x_A y_A]_\mathsf{A} = f[y_A]_\mathsf{A} \oplus g[x_A]_\mathsf{A} \oplus [w]_\mathsf{A} \oplus fg$.
2. Symmetrically the parties compute $[x_B y_B]_\mathsf{B}$.
3. The parties ask $\mathcal{F}_{\text{DEAL}}$ for a random authenticated OT $[u_0]_\mathsf{A}, [u_1]_\mathsf{A}, [c]_\mathsf{B}, [w]_\mathsf{B}$ with $w = u_c$.
   They also ask for an authenticated bit $[r_A]_\mathsf{A}$.
   Now B reveals $[d]_\mathsf{B} = [c]_\mathsf{B} \oplus [y_B]_\mathsf{B}$.
   A reveals $[f]_\mathsf{A} = [u_0]_\mathsf{A} \oplus [u_1]_\mathsf{A} \oplus [x_A]_\mathsf{A}$ and $[g]_\mathsf{A} = [r_A]_\mathsf{A} \oplus [u_0]_\mathsf{A} \oplus d[x_A]_\mathsf{A}$.
   Compute $[s_B]_\mathsf{B} = [w]_\mathsf{B} \oplus f[c]_\mathsf{B} \oplus g$. Note that at this point $[s_B]_\mathsf{B} = [r_A \oplus x_A y_B]_\mathsf{B}$.
4. Symmetrically the parties compute $[s_A]_\mathsf{A} = [r_B \oplus x_B y_A]_\mathsf{A}$.

A and B compute $[z_A]_\mathsf{A} = [r_A]_\mathsf{A} \oplus [s_A]_\mathsf{A} \oplus [x_A y_A]_\mathsf{A}$ and $[z_B]_\mathsf{B} = [r_B]_\mathsf{B} \oplus [s_B]_\mathsf{B} \oplus [x_B y_B]_\mathsf{B}$ and let $[z] = [z_A | z_B]$.

Output

The parties retrieve $[x] = [x_A | x_B]$. If A is to learn x, B reveals x_B. If B is to learn x, A reveals x_A.

Figure 4.4: Protocol for $\mathcal{F}_{\text{2PC}}$ in the $\mathcal{F}_{\text{DEAL}}$-hybrid model

an attacker A:

Global key: *No change.*

MAC query II: If A outputs a query $(\texttt{mac}, b, l, M)$, where $b \in \{0,1\}$ and l is a label which A did not use before and $M \in \{0,1\}^\psi$, let $K = M \oplus b\Delta$ and store (l, K, b).

Break query II: If A outputs a query $(\texttt{break}, \Delta')$ where $\Delta' = \Delta$, then A wins the game. This query can be used only once.

We let $\text{Game}_{II,I}$ be the hybrid game with **MAC query II** and **Break query I**.

We say that an adversary A is no stronger than adversary B if A does not perform more queries than B does and the running time of A is asymptotically linear in the running time of B.

Lemma 4.1. *For any adversary $\mathsf{A}_{I,I}$ for $\mathrm{Game}_{I,I}$ there exists an adversary $\mathsf{A}_{II,I}$ for $\mathrm{Game}_{II,I}$ which is no stronger than $\mathsf{A}_{I,I}$ and which wins the game with the same probability as $\mathsf{A}_{I,I}$.*

Proof. Given an adversary $\mathsf{A}_{I,I}$ for $\mathrm{Game}_{I,I}$, consider the following adversary $\mathsf{A}_{II,I}$ for $\mathrm{Game}_{II,I}$. The adversary $\mathsf{A}_{II,I}$ passes all side information on Δ to $\mathsf{A}_{I,I}$. If $\mathsf{A}_{I,I}$ gives the output $(\mathtt{mac}, b, l)$, then $\mathsf{A}_{II,I}$ samples $M \in_{\mathrm{R}} \{0,1\}^{\psi}$, outputs $(\mathtt{mac}, b, l, M)$ to $\mathrm{Game}_{II,I}$ and returns M to $\mathsf{A}_{I,I}$. If $\mathsf{A}_{I,I}$ outputs $(\mathtt{break}, a_1, l_1, \ldots, a_p, l_p, M')$, then $\mathsf{A}_{II,I}$ outputs $(\mathtt{break}, a_1, l_1, \ldots, a_p, l_p, M')$ to $\mathrm{Game}_{II,I}$. It is easy to see that $\mathsf{A}_{II,I}$ makes the same number of queries as $\mathsf{A}_{I,I}$ and has a running time which is linear in that of $\mathsf{A}_{I,I}$, and that $\mathsf{A}_{II,I}$ wins with the same probability as $\mathsf{A}_{I,I}$. Namely, in $\mathrm{Game}_{I,I}$ the value K is uniform and $M = K \oplus b\Delta$. In $\mathrm{Game}_{II,I}$ the value M is uniform and $K = M \oplus b\Delta$. This gives the exact same distribution on (K, M). □

Lemma 4.2. *For any adversary $\mathsf{A}_{II,I}$ for $\mathrm{Game}_{II,I}$ there exists an adversary $\mathsf{A}_{II,II}$ for $\mathrm{Game}_{II,II}$ which is no stronger than $\mathsf{A}_{II,I}$ and which wins the game with the same probability as $\mathsf{A}_{II,I}$.*

Proof. Given an adversary $\mathsf{A}_{II,I}$ for $\mathrm{Game}_{II,I}$, consider the following adversary $\mathsf{A}_{II,II}$ for $\mathrm{Game}_{II,II}$. The adversary $\mathsf{A}_{II,II}$ passes any side information on Δ to $\mathsf{A}_{II,I}$. If $\mathsf{A}_{II,I}$ outputs $(\mathtt{mac}, b, l, M)$, then $\mathsf{A}_{II,II}$ outputs $(\mathtt{mac}, b, l, M)$ to $\mathrm{Game}_{II,II}$ and stores (l, M, b). If $\mathsf{A}_{II,I}$ outputs $(\mathtt{break}, a_1, l_1, \ldots, a_p, l_p, M')$, where values $(l_1, M_1, b_1), \ldots, (l_p, M_p, b_p)$ are stored, then let $M = \oplus_{i=1}^{p} a_i M_i$ and $b = \oplus_{i=1}^{p} a_i b_i$ and output $(\mathtt{break}, M \oplus M')$. For each (l_i, M_i, b_i) let K_i be the corresponding key stored by $\mathrm{Game}_{II,II}$. We have that $M_i = K_i \oplus b_i \oplus \Delta$, so if we let $K = \oplus_{i=1}^{p} a_i K_i$, then $M = K \oplus b\Delta$. Assume that $\mathsf{A}_{II,I}$ would win $\mathrm{Game}_{II,I}$, i.e., $M' = K \oplus (1 \oplus b)\Delta$. This implies that $M \oplus M' = K \oplus b\Delta \oplus K \oplus (1 \oplus b)\Delta = \Delta$, which means that $\mathsf{A}_{II,II}$ wins $\mathrm{Game}_{II,II}$. □

Consider then the following game Game_{II} played by an attacker A:

Global key: *No change.*

MAC query: *No MAC queries are allowed.*

Break query II: *No change.*

Lemma 4.3. *For any adversary $\mathsf{A}_{II,II}$ for $\mathrm{Game}_{II,II}$ there exists an adversary A_{II} for Game_{II} which is no stronger than $\mathsf{A}_{II,II}$ and which wins the game with the same probability as $\mathsf{A}_{II,II}$.*

Proof. Let $\mathsf{A}_{II} = \mathsf{A}_{II,II}$. The game Game_{II} simply ignores the MAC queries, and it can easily be seen that they have no effect on the winning probability, so the winning probability stays the same. □

Corrollary 4.1. *For any adversary $\mathsf{A}_{I,I}$ for $\mathrm{Game}_{I,I}$ there exists an adversary A_{II} for Game_{II} which is no stronger than $\mathsf{A}_{I,I}$ and which wins the game with the same probability as $\mathsf{A}_{I,I}$.*

Why the global key queries?

The $\mathcal{F}_{\text{DEAL}}$ functionality (Figure 4.3) allows the adversary to guess the value of the global key, and it informs it if its guess is correct. This is needed for technical reasons: When $\mathcal{F}_{\text{DEAL}}$ is proved UC secure, the environment has access to either $\mathcal{F}_{\text{DEAL}}$ or the protocol implementing $\mathcal{F}_{\text{DEAL}}$. In both cases the environment learns the global keys Δ_A and Δ_B. In particular, the environment learns Δ_A even if B is honest. This requires us to prove the sub-protocol for $\mathcal{F}_{\text{DEAL}}$ secure to an adversary knowing Δ_A even if B is honest: to be able to do this, the simulator needs to recognize Δ_A if it sees it—hence the global key queries. Note, however, that in the context where we use $\mathcal{F}_{\text{DEAL}}$ (Figure 4.4), the environment does *not* learn the global key Δ_A when B is honest: A corrupted A only sees MACs on one bit using the same local key, so all MACs are uniformly random in the view of a corrupted A, and B never makes the local keys public.

Amortized MAC checks

In the protocol of Figure 4.4, there is no need to send MACs and check them every time we do a "reveal". In fact, it is straightforward to verify that before an **Output** command is executed, the protocol is perfectly secure even if the MACs are not checked. Notice then that a keyholder checks a MAC M_x on a bit x by computing $M'_x = K_x \oplus x\Delta$ and comparing M'_x to the M_x which was sent along with x. These equality checks can be deferred and amortized. Initially the MAC holder, e.g. A, sets $N = 0^\psi$ and the key holder, e.g. B, sets $N' = 0^\psi$. As long as no **Output** command is executed, when A reveals x she updates $N \leftarrow G(N, H(M_x))$ for the MAC M_x she should have sent along with x, and B updates $N' \leftarrow G(N', H(M'_x))$. Before executing an **Output**, A sends N to B who aborts if $N \neq N'$. Security of this check is easily proved in the random oracle model. The optimization brings the communication complexity of the protocol down from $O(\psi|C|)$ to $O(|C| + o\psi)$, where o is the number of rounds in which outputs are opened. For a circuit of depth $O(|C|/\psi)$, the communication is $O(|C|)$.

Implementing $\mathcal{F}_{\text{Deal}}$

In the following sections we show how to implement $\mathcal{F}_{\text{DEAL}}$. In Section 4.2 we describe how to implement the **Initialize** command and the **Authenticated Bit** commands providing aBits. In Section 4.4 we show how to extend with the **Authenticated OT** commands, by showing how to implement many aOTs from many aBits. In Section 4.3 we then show how to extend with the **Authenticated local AND** commands, by showing how to implement many aANDs from many aBits. We describe the extensions separately, but since they all maintain the value of the global keys, they will produce aANDs and aOTs with the same keys as the aBits used, giving an implementation of $\mathcal{F}_{\text{DEAL}}$. We name the functionalities that provides aBits, aOTs and aANDs $\mathcal{F}_{\text{ABIT}}$, $\mathcal{F}_{\text{AOT}}$ and $\mathcal{F}_{\text{AAND}}$ respectively. These functionalities will all provides large *batches* of their respective primitive. I.e. each of the functionalities take a parameter ℓ and produces ℓ aBits, aOTs or aANDs as one batch. The implementation of

$\mathcal{F}_{\text{DEAL}}$ is taken to run each of the underlying functionalities when initialized, and then draw from these batches to implement its commands. We note that this imposes a restriction on how many times we can use each of $\mathcal{F}_{\text{DEAL}}$'s commands. However, this is not a problem as long as we have an upper bound on the number of aBits, aANDs and aOTs we will need. For our concrete use of $\mathcal{F}_{\text{DEAL}}$ in Figure 4.4 this bound is easily computed from the size of circuit A and B wants to compute.

4.2 Bit Authentication

We describe how to implement the **Initialize** and **Authenticated Bit** commands of the $\mathcal{F}_{\text{DEAL}}$ functionality. I.e. we show how to provide oblivious authentication of random bits (aBits). As the command is completely symmetric for A and B we will just describe how to provide authenticated bits to A. As described above an aBit outputs to A a bit $x \in_{\text{R}} \{0,1\}$ and MAC $M_x \in_{\text{R}} \{0,1\}^\psi$ and to B a key $K_x \in_{\text{R}} \{0,1\}^\psi$, so that $M_x = K_x \oplus x\Delta_A$. Here $\Delta_A \in_{\text{R}} \{0,1\}^\psi$ output to B in the initialization, is the global key used to authenticate A's bits in each aBit. In the short hand notation of Figure 4.1 an aBit simply outputs $[x]_A = (x, M_x, K_x)$.

To implement the functionality $\mathcal{F}_{\text{ABIT}}$ providing aBits we simply notice that aBits corresponds exactly to the Δ-ROTs we implemented in Chapter 3. Namely a Δ-ROT is a random OT that outputs to A a choice bit b and a message N_b and to B a message N_0 so that $N_b = N_0 \oplus b\Delta$ for a global value Δ known by B. Thus if we let $x = b$, $M_x = N_b$, $K_x = N_0$ and $\Delta_A = \Delta$ we have exactly the aBit described above.

To be compatible with notation used in this chapter in Figure 4.5 we rephrase $\mathcal{F}_{\Delta\text{-ROT}}$ from Chapter 3, as the functionality $\mathcal{F}_{\text{ABIT}}$ providing ℓ aBits with keys of length ψ. There is only one significant difference between the $\mathcal{F}_{\Delta\text{-ROT}}$ and $\mathcal{F}_{\text{ABIT}}$ functionalities, namely the global key queries, which were not present the $\mathcal{F}_{\Delta\text{-ROT}}$ functionality. However, since we can clearly use the functionality *without* the global key queries to implement the functionality *with*, we can also use the protocol for $\mathcal{F}_{\Delta\text{-ROT}}$ from Chapter 3 to implement $\mathcal{F}_{\text{ABIT}}$. We get the following theorem simply restating Cor. 3.2 of Chapter 3.

Theorem 4.2. *Let ψ denote the security parameter and let $\ell = \text{poly}(\psi)$. The functionality $\mathcal{F}_{\text{ABIT}}(\ell, \psi)$ can be reduced to $(\mathcal{F}_{\text{OT}}(\frac{44}{3}\psi, \psi), \mathcal{F}_{\text{EQ}}(\psi))$. The communication is $O(\psi\ell + \psi^2)$ and the work is $O(\psi^2\ell)$.*

4.3 Authenticated local AND

In this section we show how to implement the functionality $\mathcal{F}_{\text{AAND}}$ providing authenticated local ANDs (aANDs). I.e. how to implement the dealer functionality when it outputs $[x]_{\mathsf{A}}, [y]_{\mathsf{A}}, [z]_{\mathsf{A}}$ with $z = xy$. As aAND for B is symmetric, we only present how to construct $\mathcal{F}_{\text{AAND}}$ providing aANDs to A.

We first construct a leaky version of $\mathcal{F}_{\text{AAND}}$, described in Figure 4.6. The leaky $\mathcal{F}_{\text{AAND}}$ functionality is leaky in the sense that B may learn some of the

Honest Parties

On input `start` from both A and B the functionality does the following.

1. The functionality samples $\Delta_A \in_{\mathrm{R}} \{0,1\}^{\psi}$ and outputs it to A.
2. For all $i \in [\ell]$ the functionality samples $x_i \in_{\mathrm{R}} \{0,1\}$ and $K_{x_i} \in_{\mathrm{R}} \{0,1\}^{\psi}$ and lets $M_{x_i} = K_{x_i} \oplus x_i \Delta_A$.
3. The functionality outputs $([x_i]_A = (x_i, M_{x_i}, K_{x_i}))_{i\in[\ell]}$. I.e. for each $i \in [\ell]$ it outputs (M_{x_i}, x_i) to A and K_{x_i} to B.

Corrupted Parties

1. If A is corrupted, the functionality waits to give output till it receives the message $(\hat{M}_{\hat{x}_i}, \hat{x}_i)_{i\in[\ell]}$ from A, where $\hat{M}_{\hat{x}_i} \in \{0,1\}^{\psi}$ and $\hat{x}_i \in \{0,1\}$. The functionality then sets $x_i = \hat{x}_i$, $M_{x_i} = \hat{M}_{\hat{x}_i}$ and $K_{x_i} = M_{x_i} \oplus x_i \Delta_A$ and outputs as described above (with Δ_A sampled as above).
2. If B is corrupted, the functionality waits to give output till it receives the message $(\hat{\Delta}_A, (\hat{K}_{x_i})_{i\in[\ell]})$ from A, where $\hat{\Delta}_A, \hat{K}_{x_i} \in \{0,1\}^{\psi}$. The functionality then sets $\Delta_A = \hat{\Delta}_A$, $K_{x_i} = \hat{K_{x_i}}$ and $M_{x_i} = K_{x_i} \oplus x_i \Delta_A$ and outputs as described above (with x_i sampled as above).

Global Key Queries

If the adversary inputs $(\mathtt{guess}, \hat{\Delta})$ to the functionality, the functionality outputs back `correct` if $\hat{\Delta} = \Delta$ and `incorrect` otherwise.

Figure 4.5: The $\mathcal{F}_{\mathrm{ABIT}}(\ell, \tau)$ functionality

Honest Parties

For $i = 1, \ldots, \ell$, the functionality outputs random $[x_i]_{\mathsf{A}}, [y_i]_{\mathsf{A}}, [z_i]_{\mathsf{A}}$ with $z_i = x_i y_i$.

Corrupted Parties

1. If A is corrupted she gets to choose all her random values, similar to how she gets to choose them in $\mathcal{F}_{\mathrm{ABIT}}$.
2. If B is corrupted he gets to choose all his random values, including the global key Δ_A. Also, he may, at any point prior to output being delivered to A, input (i, g_i) to the functionality in order to try to guess x_i. If $g_i \neq x_i$ the functionality will output `fail` to A and terminate. Otherwise the functionality proceeds as if nothing has happened and B will know the guess was correct. He may make as many guesses as he desires.

Global Key Queries

The adversary can input Δ and will be told if $\Delta = \Delta_A$.

Figure 4.6: The functionality leaky $\mathcal{F}_{\mathrm{AAND}}(\ell)$ for ℓ Leaky Authenticated local ANDs.

values x_i: a corrupted B is allowed to try to guess the x_i bits, but if the guess is wrong the functionality aborts revealing that B cheated. This means that if the functionality does not abort, with very high probability B only tried to guess a few bits.

The intuition behind the protocol for leaky $\mathcal{F}_{\mathrm{AAND}}$, described in Figure 4.7, is to let A compute the AND locally and then authenticate the result. A and

B then perform some computation on the keys and MACs, in a way so that A will be able to guess B's result only if she behaved honestly during the protocol: A behaved honestly (sent $d = z \oplus r$) iff she knows $W_0 = (K_x||K_z)$ or $W_1 = (K_x \oplus \Delta_A||K_y \oplus K_z)$. In fact, she knows W_x. As an example, if $x = 0$ and A is honest, then $z = 0$, so she knows $M_x = K_x$ and $M_z = K_z$. Had she cheated, she would know $M_z = K_z \oplus \Delta_A$ instead of K_z. B checks that A knows W_0 or W_1 by sending her $H(W_0) \oplus H(W_1)$ and ask her to return $H(W_0)$. This, however, allows B to send $H(W_0) \oplus H(W_1) \oplus E$ for an error term $E \neq 0^\psi$. The returned value would be $H(W_0) \oplus xE$. To prevent this attack, we use the $\mathcal{F}_{\text{EQ}}$ functionality to compare the values instead. If B uses $E \neq 0^\psi$, he must now guess x to pass the protocol. However, B still may use this technique to guess a few x bits. The prove of the following theorem.

The protocol runs ℓ times in parallel. Here described for a single leaky authenticated local AND:

1. A and B ask the dealer for $[x]_\mathsf{A}, [y]_\mathsf{A}, [r]_\mathsf{A}$.
2. A computes $z = xy$ and announces $d = z \oplus r$.
3. The parties compute $[z]_\mathsf{A} = [r]_\mathsf{A} \oplus d$.
4. B sends $U = H(K_x||K_z) \oplus H(K_x \oplus \Delta_A||K_y \oplus K_z)$ to A.
5. If $x = 0$, then A lets $V = H(M_x||M_z)$. If $x = 1$, then A lets $V = U \oplus H(M_x||M_y \oplus M_z)$.
6. A and B call the $\mathcal{F}_{\text{EQ}}$ functionality, with inputs V and $H(K_x||K_z)$ respectively. All the ℓ calls to $\mathcal{F}_{\text{EQ}}$ are handled using a single call to $\mathcal{F}_{\text{EQ}}(\ell\psi)$.
7. If the strings were not different, the parties output $[x]_\mathsf{A}, [y]_\mathsf{A}, [z]_\mathsf{A}$.

Figure 4.7: Protocol for leaky authenticated local AND

Theorem 4.3. *The protocol in Figure 4.7 securely implements leaky $\mathcal{F}_{\text{AAND}}(\ell)$ in the $(\mathcal{F}_{\text{ABIT}}(3\ell, \psi), \mathcal{F}_{\text{EQ}}(\ell\psi))$-hybrid model.*

The simulator answers global key queries to the dealer by doing the identical global key queries on the leaky $\mathcal{F}_{\text{AAND}}(\ell)$ functionality and returning the reply. This gives a perfect simulation of these queries, and we ignore them below.

Notice that for honest A and B correctness of the protocol follows immediately from correctness of the $\mathcal{F}_{\text{ABIT}}$ functionality. Thus we prove Theorem 4.3 by proving security separately against corrupted A and B respectively. We do this in Lemma 4.4 and 4.5.

Lemma 4.4. *The protocol in Figure 4.7 securely implements the leaky $\mathcal{F}_{\text{AAND}}$ functionality against corrupted* A.

Proof. We first focus on the simulation of the protocol before outputs are given to the environment. Notice that before outputs are given to the environment, the global key Δ_A is uniformly random to the environment, as long as B is honest.

We consider the case of a corrupt sender A^* running the above protocol against a simulator S for honest B.

1. S receives A^*'s input $(M_x, x), (M_y, y), (M_r, r)$ to the dealer, and the bit $d \in_{\text{R}} \{0, 1\}$.

2. S samples a random $U \in_R \{0,1\}^{2\psi}$ and sends it to A^*. Then S reads A^*'s input to the $\mathcal{F}_{\text{EQ}}$ functionality, $\overline{V}$. If

$$\overline{V} \neq (1-x)H(M_x, M_z) \oplus x(U \oplus H(M_x, M_y \oplus M_z))$$

or

$$d \oplus y \neq xy \ ,$$

S outputs abort. Otherwise, it inputs $(x, y, z, M_x, M_y, M_z = M_r)$ to the leaky $\mathcal{F}_{\text{AAND}}$ functionality.

The first difference between the real protocol and the simulation is that

$$U = H(K_x, K_z) \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z)$$

in the real protocol and U is uniformly random in the simulation. Since H modeled as a random oracle, this is perfectly indistinguishable to the adversary until it queries on both (K_x, K_z) and $(K_x \oplus \Delta_A, K_y \oplus K_z)$. Since Δ_A is uniformly random to the environment and the adversary during the protocol, this will happen with negligible probability during the protocol. We later return to how we simulate after outputs are given to the environment.

The other difference between the protocol and the simulation is that the simulation always aborts if $z \neq xy$. Assume now that A^* manages, in the real protocol, to make the protocol continue with $z = xy \oplus 1$. If $x = 0$, this means that A^* queried H on

$$(K_x, K_z) = (M_x, M_z \oplus \Delta_A).$$

Since S knows the outputs of A^*, which include M_z, and sees the input $M_z \oplus \Delta_A$ to H, if A^* queries the oracle on $(K_x, K_z) = (M_x, M_z \oplus \Delta_A)$, S can compute Δ_A. If $x = 1$ then A^* must have queried the oracle on

$$(K_x \oplus \Delta_A, K_y \oplus K_z) = (M_x, M_y \oplus M_z \oplus \Delta_A) \ ,$$

which again would allow S to compute Δ_A. Therefore, in both cases we can use such an A^* to compute the global key Δ_A and, given that all of A^*'s inputs are independent of Δ_A during the protocol, this happens only with negligible probability.

Consider now the case after the environment is given outputs. These outputs include Δ_A. It might seem that there is nothing more to simulate after outputs are given, but recall that H is a random oracle simulated by S and that the environment might keep querying H. Our concern is that U is uniformly random in the simulation and

$$U = H(K_x, K_z) \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z)$$

in the real protocol. We handle this as follows. Each time the environment queries H on an input of the form $(Q_1, Q_2) \in \{0,1\}^{2\psi}$, go over all previous queries (Q_3, Q_4) of this form and let $\Delta = Q_1 \oplus Q_3$. Then do a global key query to $\mathcal{F}_{\text{ABIT}}(3\ell, \psi)$ to determine if $\Delta = \Delta_A$. If S learns Δ_A this way, she proceeds

as follows. Note that since A^* is corrupted, S knows all outputs to A^*, i.e., S knows all MACs M and all bits b. If $b = 0$, then S also knows the key, as $K = M$ when $b = 0$. If $b = 1$, S computes the key as $K = M \oplus \Delta_A$. So, when S learns Δ_A, she at the same time learns all keys. Then for each U she simply programs the RO such that

$$U = H(K_x, K_z) \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z) \ .$$

This is possible as S learns Δ_A no later than when the environment queries on two pairs of inputs of the form $(Q_1, Q_2) = (K_x, K_z)$ and $(Q_3, Q_4) = (K_x \oplus \Delta_A, K_y \oplus K_z)$. So, when S learns Δ_A, either $H(K_x, K_z)$ or $H(K_x \oplus \Delta_A, K_y \oplus K_z)$ is still undefined. If it is $H(K_x, K_z)$, say, which is undefined, S simply programs the RO so that

$$H(K_x, K_z) = U \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z) \ .$$

□

Lemma 4.5. *The protocol described in Figure 4.7 securely implements the leaky $\mathcal{F}_{\text{AAND}}$ functionality against corrupted B.*

Proof. We consider the case of a corrupt B^* running the above protocol against a simulator S. The simulation runs as follows.

1. The simulation starts by S getting B^*'s input to the dealer K_x, K_y, K_r and Δ_A.

2. The simulator samples a random $d \in_R \{0, 1\}$, sends it to B^* and computes $K_z = K_r \oplus d\Delta_A$.

3. S receives $\overline{U}$ from B^*, and reads $\overline{V}$, B^*'s input to the equality functionality.

4. If
$$\overline{U} = H(K_x, K_z) \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z)$$
and
$$\overline{V} = H(K_x, K_z) \ ,$$
S inputs (K_x, K_y, K_z) to the leaky $\mathcal{F}_{\text{AAND}}$ functionality and completes the protocol (in this case B^* is behaving honestly). Otherwise, if
$$\overline{U} \neq H(K_x, K_z) \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z)$$
and
$$\overline{V} = H(K_x, K_z)$$
or
$$\overline{V} = \overline{U} \oplus H(K_x \oplus \Delta_A, K_z \oplus K_z) \ ,$$
S inputs $g = 0$ or $g = 1$ resp. to the leaky $\mathcal{F}_{\text{AAND}}$ functionality as a guess for the bit x. If the functionality output `fail`, output `fail` and abort, and otherwise complete the protocol.

The simulation is perfect: the view of B^* consists only of the bit d, and whether or not A aborts. Here d is uniformly distributed both in the real world and in the simulation, and the protocol aborts based on the same event in the real and in the simulated world.

□

Combining Lemma 4.4 and 4.5, we get the proof of Theorem 4.3.

We now handle a few guessed x bits by random bucketing and a straightforward combiner. In doing this efficiently, it is central that the protocol was constructed such that only x could leak. Had B been able to get information on both x and y we would have had to do the amplification twice.

The protocol is parametrized by positive integers B and ℓ.

1. A and B call leaky $\mathcal{F}_{\text{AAND}}(\ell')$ with $\ell' = B\ell$. If the call to leaky $\mathcal{F}_{\text{AAND}}$ aborts, this protocol aborts. Otherwise, let $\{[x_i]_\mathsf{A}, [y_i]_\mathsf{A}, [z_i]_\mathsf{A}\}_{i=1}^{\ell'}$ be the outputs.
2. A picks a B-wise independent permutation π on $\{1, \ldots, \ell'\}$ and sends it to B. For $j = 0, \ldots, \ell - 1$, the B triples
$$\{[x_{\pi(i)}]_\mathsf{A}, [y_{\pi(i)}]_\mathsf{A}, [z_{\pi(i)}]_\mathsf{A}\}_{i=jB+1}^{jB+B}$$
are defined to be in the j'th bucket.
3. A and B combine B triples in each bucket. Here we describe how to combine two triples: Let be them $[x^1]_\mathsf{A}, [y^1]_\mathsf{A}, [z^1]_\mathsf{A}$ and $[x^2]_\mathsf{A}, [y^2]_\mathsf{A}, [z^2]_\mathsf{A}$. Let the result be $[x]_\mathsf{A}, [y]_\mathsf{A}, [z]_\mathsf{A}$.
   (a) A reveals $d = y^1 \oplus y^2$.
   (b) A and B compute
$$[x]_\mathsf{A} = [x^1]_\mathsf{A} \oplus [x^2]_\mathsf{A} \quad , \quad [y]_\mathsf{A} = [y^1]_\mathsf{A} \ ,$$
   and
$$[z]_\mathsf{A} = [z^1]_\mathsf{A} \oplus [z^2]_\mathsf{A} \oplus d[x^2]_\mathsf{A} \ .$$
   To combine all B triples in a bucket, just iterate by taking the result and combine it with the next element in the bucket.

Figure 4.8: From Leaky Authenticated local ANDs to Authenticated local ANDs

We start by producing $B\ell$ leaky aANDs. Then we randomly distribute the $B\ell$ leaky aANDs into ℓ buckets of size B. Finally we combine the leaky aANDs in each bucket into one aAND which is non-leaky if at least one leaky aAND in the bucket was not leaky. The protocol is described in Figure 4.8. We prove the following theorem.

Theorem 4.4. *Let $\mathcal{F}_{\text{AAND}}(\ell)$ denote the functionality providing ℓ non-leaky aANDs as in $\mathcal{F}_{\text{DEAL}}$. For $B \geq \frac{\sigma}{1+\log_2(\ell)} + 1$, the protocol in Figure 4.8 securely implements $\mathcal{F}_{\text{AAND}}(\ell)$ in the leaky $\mathcal{F}_{\text{AAND}}(B\ell)$-hybrid model with statistical security parameter σ.*

Proof. The simulator answers global key queries to leaky $\mathcal{F}_{\text{AAND}}(B\ell)$ by doing the identical global key queries on the ideal functionality $\mathcal{F}_{\text{AAND}}(\ell)$ and returning the reply. This gives a perfect simulation of these queries, and we ignore them below.

It is easy to check that the protocol is correct and secure if both parties are honest or if A is corrupted.

What remains is to show that, even if B is corrupted and tries to guess some x's from the leaky $\mathcal{F}_{\text{AAND}}$ functionality, the overall protocol is secure.

We argue this in two steps. We first argue that the probability that B learns the x-bit for all triples in the same bucket is negligible. We then argue that when all buckets contain at least one triple for which x is unknown to B, then the protocol can be simulated given leaky $\mathcal{F}_{\text{AAND}}(B\ell)$.

Call each of the triples a *ball* and call a ball *leaky* if B learned the x bit of the ball in the call to the leaky $\mathcal{F}_{\text{AAND}}(\ell')$ functionality. Let γ denote the number of leaky balls.

For B of the leaky balls to end up in the same bucket, there must be a subset S of balls with $|S| = B$ consisting of only leaky balls and a bucket i such that all the balls in S end up in i.

We first fix S and i and compute the probability that all balls in S end up in i. The probability that the first ball ends up in i is $\frac{B}{B\ell}$. The probability that the second balls ends up in i given that the first ball is in i is $\frac{B-1}{B\ell-1}$, and so on. We get a probability of

$$\frac{B}{B\ell} \cdot \frac{B-1}{B\ell-1} \cdots \frac{1}{B\ell-B+1} = \binom{B\ell}{B}^{-1}$$

that S ends up in i.

There are $\binom{\gamma}{B}$ subsets S of size B consisting of only leaky balls and there are ℓ buckets, so by a union bound the probability that any bucket is filled by leaky balls is upper bounded by

$$\binom{\gamma}{B}\ell\binom{B\ell}{B}^{-1} .$$

This is assuming that there are exactly γ leaky balls. Note then that the probability of the protocol not aborting when there are γ leaky balls is $2^{-\gamma}$. Namely, for each bit x that B tries to guess, he is caught with probability $\frac{1}{2}$. So, the probability that B undetected can introduce γ leaky balls and have them end up in the same bucket is upper bounded by

$$\alpha(\gamma, \ell, B) = 2^{-\gamma}\binom{\gamma}{B}\ell\binom{B\ell}{B}^{-1} .$$

It is easy to see that

$$\frac{\alpha(\gamma+1, \ell, B)}{\alpha(\gamma, \ell, B)} = \frac{\gamma+1}{2(\gamma+1-B)}.$$

So, $\alpha(\gamma+1, \ell, B)/\alpha(\gamma, \ell, B) > 1$ iff $\gamma < 2B-1$, hence $\alpha(\gamma, \ell, B)$ is maximized in γ at $\gamma = 2B-1$. If we let $\alpha'(B, \ell) = \alpha(2B-1, \ell, B)$ it follows that the success probability of the adversary is at most

$$\alpha'(B, \ell) = 2^{-2B+1}\ell\frac{(2B-1)!(B\ell-B)!}{(B-1)!(B\ell)!} .$$

Writing out the product $\frac{(2B-1)!(B\ell-B)!}{(B-1)!(B\ell)!}$ it is fairly easy to see that for $2 \leq B < \ell$ we have that

$$\frac{(2B-1)!(B\ell-B)!}{(B-1)!(B\ell)!} < \frac{(2B)^B}{(B\ell)^B},$$

so

$$\alpha'(B,\ell) \leq 2^{-2B+1}\ell\frac{(2B)^B}{(B\ell)^B} = (2\ell)^{1-B} = 2^{(\log_2(\ell)+1)(1-B)}.$$

Thus for $B \geq \frac{\sigma}{1+\log_2(\ell)} + 1$ we have $\alpha'(B,\ell) \leq 2^{-\sigma}$ is negligible in σ.

We now prove that assuming each bucket has one non-leaky triple the protocol is secure even for a corrupted B^*.

We look only at the case of two triples, $[x^1]_\mathsf{A}$,$[y^1]_\mathsf{A}$,$[z^1]_\mathsf{A}$ and $[x^2]_\mathsf{A}$,$[y^2]_\mathsf{A}$,$[z^2]_\mathsf{A}$ being combined into $[x]_\mathsf{A}, [y]_\mathsf{A}, [z]_\mathsf{A}$. It is easy to see why this is sufficient: Consider the iterative way we combine the B triples of a bucket. At each step we combine two triples where one may be the result of previous combinations. Thus if a combination of two triples, involving a non-leaky triple, results in a non-leaky triple, the subsequent combinations involving that result will all result in a non-leaky triple.

In the real world a corrupted B^* will input keys K_{x^1}, K_{y^1}, K_{z^1} and K_{x^2}, K_{y^2}, K_{z^2} and Δ_A, and possibly some guesses at the x-bits to the leaky $\mathcal{F}_{\text{AAND}}$ functionality. Then B^* will see $d = y^1 \oplus y^2$ and $M_d = (K_{y^1} \oplus K_{y^2}) \oplus d\Delta_A$ and A will output $x = x^1 \oplus x^2$, $y = y^1$, $z = z^1 \oplus z^2 \oplus dx^2$ and $M_x = (K_{x^1} \oplus K_{x^2}) \oplus x\Delta_A$, $M_y = K_{y^1} \oplus y\Delta_A$, $M_z = (K_{z^1} \oplus K_{z^2} \oplus dK_{x^2}) \oplus z\Delta_A$ to the environment.

Consider then a simulator Sim running against B^* and using an $\mathcal{F}_{\text{AAND}}$ functionality. In the first step Sim gets all B^*'s keys like in the real world. If B^* submits a guess (i, g_i) Sim simply outputs `fail` and terminates with probability $\frac{1}{2}$. To simulate revealing d, Sim samples $d \in_\text{R} \{0,1\}$, sets $M_d = K_{y^1} \oplus K_{y^2} \oplus d\Delta_A$ and sends d and M_d to B^*. Sim then forms the keys $K_x = K_{x^1} \oplus K_{x^2}$, $K_y = K_{y^1}$ and $K_z = K_{z^1} \oplus K_{z^2} \oplus dK_{x^2}$ and inputs them to the $\mathcal{F}_{\text{AAND}}$ functionality on behalf of B^*. Finally the $\mathcal{F}_{\text{AAND}}$ functionality will output random x, y and $z = xy$ and $M_x = K_x \oplus x\Delta_A$, $M_y = K_y \oplus y\Delta_A$, $M_z = K_z \oplus z\Delta_A$.

We have already argued that the probability of B^* guessing one of the x-bits is exactly $\frac{1}{2}$, so Sim terminates the protocol with the exact same probability as the leaky $\mathcal{F}_{\text{AAND}}$ functionality in the real world. Notice then that, given the assumption that B^* at most guesses one of the x-bits, all bits d, x and y are uniformly random to the environment both in the real world and in the simulation. Thus because Sim can form the keys K_x, K_y and K_z to the $\mathcal{F}_{\text{AAND}}$ functionality exactly as they would be in the real world the simulation will be perfect.

□

4.4 Authenticated Oblivious Transfer

In this section we show how to implement the functionality $\mathcal{F}_{\text{AOT}}$ providing authenticated oblivious transfers (aOTs). I.e., we implement the extended $\mathcal{F}_{\text{DEAL}}$ functionality when it outputs $[x_0]_\mathsf{A}, [x_1]_\mathsf{A}, [c]_\mathsf{B}, [z]_\mathsf{B}$ with $z = c(x_0 \oplus x_1) \oplus x_0 = x_c$.

Because of symmetry we only show the construction of $\mathcal{F}_{\text{AOT}}$ with A as sender and B as receiver.

Honest Parties

For $i = 1, \ldots, \ell$, the functionality outputs random $[x_0^i]_{\mathsf{A}}, [x_1^i]_{\mathsf{A}}, [c^i]_{\mathsf{B}}, [z^i]_{\mathsf{B}}$ with $z^i = c^i(x_0^i \oplus x_1^i) \oplus x_0^i$.

Corrupted Parties

1. If B is corrupted he gets to choose all his random values.
2. If A is corrupted she gets to choose all her random values. Also, she may, at any point before B received his outputs, input (i, g^i) to the functionality in order to try to guess c^i. If $c^i \neq g^i$ the functionality will output `fail` and terminate. Otherwise the functionality proceeds as if nothing has happened and A will know the guess was correct. She may input as many guesses as she desires.

Global Key Queries

The adversary can at any point input (A, Δ) and will be returned whether $\Delta = \Delta_B$. And it can at any point input (B, Δ) and will be returned whether $\Delta = \Delta_A$.

Figure 4.9: The Leaky Authenticated OT functionality $\mathcal{F}_{\text{AOT}}(\ell)$

We go via a leaky version of $\mathcal{F}_{\text{AOT}}$, described in Figure 4.9. The leaky $\mathcal{F}_{\text{AOT}}$ functionality is leaky in the sense that choice bits may leak when A is corrupted: a corrupted A is allowed to make guesses on choice bits, but if the guess is wrong the functionality aborts revealing that A is cheating. This means that if the functionality does not abort, with very high probability A only tried to guess a few choice bits.

The protocol to construct leaky $\mathcal{F}_{\text{AOT}}$ (described in Figure 4.10) proceeds as follows: First A and B get $[x_0]_{\mathsf{A}}, [x_1]_{\mathsf{A}}$ (A's messages), $[c]_{\mathsf{B}}$ (B's choice bit) and $[r]_{\mathsf{B}}$. Then A transfers the message $z = x_c$ to B in the following way: B knows the MAC for his choice bit M_c, while A knows the two keys K_c and Δ_B. This allows A to compute the two possible MACs $(K_c, K_c \oplus \Delta_B)$ respectively for the case of $c = 0$ and $c = 1$. Hashing these values leaves A with two uncorrelated strings $H(K_c)$ and $H(K_c \oplus \Delta_B)$, one of which B can compute as $H(M_c)$. These values can be used as a one-time pad for A's bits x_0, x_1 (and some other values as described later), and B can retrieve x_c and announce the difference $d = x_c \oplus r$ and therefore compute the output $[z]_{\mathsf{B}} = [r]_{\mathsf{B}} \oplus d$.

In order to check if A is transmitting the correct bits x_0, x_1, she will transfer the respective MACs together with the bits: as B is supposed to learn x_c, revealing the MAC on this bit does not introduce any insecurity. However, A can now mount a selective failure attack: A can check if B's choice bit c is equal to, e.g., 0 by sending x_0 with the correct MAC and x_1 together with an incorrect MAC. Now if $c = 0$ B only sees the valid MAC and continues the protocol, while if $c = 1$ B aborts because of the incorrect MAC. A similar attack can be mounted to check if $c = 1$.

On the other hand, if B is corrupted, he could be announcing the wrong value d. In particular, A needs to check that the authenticated bit $[z]_{\mathsf{B}}$ is equal to x_c without learning c. In order to do this, we have A choosing two random

The protocol runs ℓ times in parallel, here described for a single leaky authenticated $\mathcal{F}_{\text{OT}}$.

1. A and B get $[x_0]_\mathsf{A}, [x_1]_\mathsf{A}, [c]_\mathsf{B}, [r]_\mathsf{B}$ from the dealer.
2. Let $[x_0]_\mathsf{A} = (x_0, M_{x_0}, K_{x_0}), [x_1]_\mathsf{A} = (x_1, M_{x_1}, K_{x_1}), [c]_\mathsf{B} = (c, M_c, K_c), [r]_\mathsf{B} = (r, M_r, K_r)$.
3. A chooses random strings $T_0, T_1 \in \{0,1\}^\psi$.
4. A sends (X_0, X_1) to B where $X_0 = H(K_c) \oplus (x_0 || M_{x_0} || T_{x_0})$ and $X_1 = H(K_c \oplus \Delta_B) \oplus (x_1 || M_{x_1} || T_{x_1})$.
5. B computes $(x_c || M_{x_c} || T_{x_c}) = X_c \oplus H(M_c)$. B aborts if $M_{x_c} \neq K_{x_c} \oplus x_c \Delta_A$. Otherwise, let $z = x_c$.
6. B announces $d = z \oplus r$ to A and the parties compute $[z]_\mathsf{B} = [r]_\mathsf{B} \oplus d$. Let $[z]_\mathsf{B} = (z, M_z, K_z)$.
7. A sends (I_0, I_1) to B where $I_0 = H(K_z) \oplus T_1$ and $I_1 = H(K_z \oplus \Delta_B) \oplus T_0$.
8. B computes $T_{1 \oplus z} = I_z \oplus H(M_z)$. *Notice that now* B *knows both* (T_0, T_1).
9. A and B both input (T_0, T_1) to $\mathcal{F}_{\text{EQ}}$. The comparisons all ℓ parallel runs are done using a single call to $\mathcal{F}_{\text{EQ}}(2\ell\psi)$.
10. If the values are the same, A and B output $[x_0]_\mathsf{A}, [x_1]_\mathsf{A}, [c]_\mathsf{B}, [z]_\mathsf{B}$.

Figure 4.10: The protocol for authenticated OT with leaky choice bit

strings T_0, T_1, and append them, respectively, to x_0, x_1 and the MACs on those bits, so that B learns T_c together with x_c. After B announces d, we can again use the MAC and the keys for z to perform a new transfer: A uses $H(K_z)$ as a one-time pad for T_1 and $H(K_z \oplus \Delta_B)$ as a one-time pad for T_0. Using M_z, the MAC on z, B can retrieve $T_{1 \oplus z}$. This means that an honest B, that sets $z = x_c$, will know both T_0 and T_1, while a dishonest B will not be able to know both values except with negligible probability. Using the $\mathcal{F}_{\text{EQ}}$ functionality A can check that B knows both values T_0, T_1. Note that we cannot simply have B openly announce these values, as this would open the possibility for new attacks on A's side. We prove the following theorem.

Theorem 4.5. *The protocol in Figure 4.10 securely implements the leaky* $\mathcal{F}_{\text{AOT}}(\ell)$ *functionality in the* $(\mathcal{F}_{\text{ABIT}}(4\ell, \psi), \mathcal{F}_{\text{EQ}}(2\ell\psi))$*-hybrid model.*

The simulator answers global key queries to the dealer by doing the identical global key queries on the leaky $\mathcal{F}_{\text{AOT}}(\ell)$ functionality and returning the reply. This gives a perfect simulation of these queries, and we ignore them below.

Notice that for honest sender and receiver correctness of the protocol follows immediately from correctness of the $\mathcal{F}_{\text{ABIT}}$ functionality. Thus we prove Theorem 4.5 by proving security separately against corrupted A and B respectively. We do this in Lemma 4.6 and 4.7.

Lemma 4.6. *The protocol in Figure 4.10 securely implements the leaky* $\mathcal{F}_{\text{AOT}}(\ell)$ *functionality against corrupted* A.

Proof. We consider the case of a corrupt sender A^* running the above protocol against a simulator S. We show how to simulate one instance.

1. First S receives A^*'s input (M_{x_0}, x_0), (M_{x_1}, x_1), K_c, K_r and Δ_B to the dealer. Then S samples a bit $y \in_\text{R} \{0,1\}$, sets $K_z = K_r \oplus y\Delta_B$ and inputs (M_{x_0}, x_0), (M_{x_1}, x_1), K_c, K_z and Δ_B to the leaky $\mathcal{F}_{\text{AOT}}$ functionality. The

functionality outputs Δ_A, (M_c, c), (M_z, z), K_{x_0} and K_{x_1} to the honest B as described in the protocol.

2. A^* outputs the message (X_0, X_1). The simulator knows Δ_B and K_c and can therefore compute

$$X_0 \oplus H(K_c) = (\overline{x}_0 || \overline{M}_{x_0} || T'_{x_0})$$

and

$$X_1 \oplus H(K_c \oplus \Delta_B) = (\overline{x}_1 || \overline{M}_{x_1} || T'_{x_1}) \ .$$

For all $j \in \{0,1\}$ S tests if $(\overline{M}_{x_j}, \overline{x}_j) = (M_{x_j}, x_j)$. If, for some j, this is not the case S inputs a guess to the leaky $\mathcal{F}_{\text{AOT}}$ functionality guessing that $c = (1-j)$. If the functionality outputs `fail` S does the same and aborts the protocol. Otherwise S proceeds by sending y to A^*. Notice that if S does not abort but does guess the choice bit c it can perfectly simulate the remaining protocol. In the following we therefore assume this is not the case.

3. Similarly S gets (I_0, I_1) from A^* and computes

$$I_0 \oplus H(K_z) = T''_1$$

and

$$I_1 \oplus H(K_z \oplus \Delta_B) = T''_0 \ .$$

4. When S receives A^*'s input (T_0, T_1) for the $\mathcal{F}_{\text{EQ}}$ functionality it first tests if $(T'_j, T''_{1 \oplus \overline{x}_j}) = (T_{\overline{x}_j}, T_{1 \oplus \overline{x}_j})$ for all $j \in \{0,1\}$. If, for some j, this is not the case S inputs a guess to the leaky $\mathcal{F}_{\text{AOT}}$ functionality guessing that $c = (1-j)$. If the functionality outputs `fail`, S outputs `fail` and aborts. If not, the simulation is over.

For analysis of the simulation we denote by F the event that for some $j \in \{0,1\}$ A^* computes values $M^*_{x_j} \in \{0,1\}^\psi$ and $x^*_j \in \{0,1\}$ so that $(M^*_{x_j}, x^*_j) \neq (M_{x_j}, x_j)$ and $M^*_{x_j} = K_{x_j} \oplus x^*_j \Delta_A$. In other words, F is the event that A^* computes a MAC on a message bit it was not supposed to know. We will now show that, assuming F does not occur, the simulation is perfectly indistinguishable from the real protocol. We then show that F only occurs with negligible probability and therefore that simulation and the real protocol are indistinguishable.

From the definition of the leaky $\mathcal{F}_{\text{AOT}}$ functionality we have that $(\overline{M}_{x_j}, \overline{x}_j) = (M_{x_j}, x_j)$ implies $\overline{M}_{x_j} = K_{x_j} \oplus x_j \Delta_A$. Given the assumption that F does not occur clearly we have that $(\overline{M}_{x_j}, \overline{x}_j) \neq (M_{x_j}, x_j)$ also implies $\overline{M}_{x_j} \neq K_{x_j} \oplus \overline{x}_j \Delta_A$. This means that S aborts in step 2 with exactly the same probability as the honest receiver would in the real protocol. Also, in the real protocol we have $y = z \oplus r$ for $r \in_{\text{R}} \{0,1\}$ thus both in the real protocol and the simulation y is distributed uniformly at random in the view of A^*.

Next in step 4 of the simulation notice that in the real protocol, if $c = j \in \{0,1\}$, an honest B would input T'_j and $T''_{1 \oplus \overline{x}_j}$ to $\mathcal{F}_{\text{EQ}}$ (sorted in the correct order). The protocol would then continue if and only if $(T'_j, T''_{1 \oplus \overline{x}_j}) = (T_{\overline{x}_j}, T_{1 \oplus \overline{x}_j})$

and abort otherwise. I.e., the real protocol would continue if and only if $(T'_j, T''_{1\oplus\overline{x}_j}) = (T_{\overline{x}_j}, T_{1\oplus\overline{x}_j})$ and $c = j$, which is exactly what happens in the simulation. Thus we have that given F does not occur, all input to A^* during the simulation is distributed exactly as in real protocol. In other words the two are perfectly indistinguishable.

Now assume F does occur, that is for some $j \in \{0,1\}$ A^* computes values $M^*_{x_j}$ and x^*_j as described above. In that case A^* could compute the global key of the honest receiver as $M^*_j \oplus M_{x_j} = \Delta_A$. However, since all inputs to A^* are independent from Δ_A (during the protocol), A^* can only guess Δ_A with negligible probability (during the protocol) and thus F can only occur with negligible probability (during the protocol). After the protocol A^*, or rather the environment, will receive outputs and learn Δ_A, but this does not change the fact that guessing Δ_A during the protocol can be done only with negligible probability. □

Lemma 4.7. *The protocol in Figure 4.10 securely implements leaky $\mathcal{F}_{\mathrm{AOT}}(\ell)$ against corrupted B.*

Proof. We consider the case of a corrupt receiver B^* running the above protocol against a simulator S. The simulation runs as follows.

1. The simulation starts by S getting B^*'s input to dealer Δ_A, (M_c, c), (M_r, r), K_{x_0} and K_{x_1}. Then S simply inputs Δ_A, (M_c, c), $M_z = M_r$, K_{x_0} and K_{x_1} to the leaky $\mathcal{F}_{\mathrm{AOT}}$ functionality. The functionality outputs z to S and Δ_B, (M_{x_0}, x_0), (M_{x_1}, x_1), K_c and K_z to the sender as described above.

2. Like the honest sender S samples random keys $T_0, T_1 \in_{\mathrm{R}} \{0,1\}^\psi$. Since S knows $M_c, K_{x_0}, K_{x_1}, \Delta_A, c$ and $z = x_c$ it can compute $X_c = H(M_c) \oplus (z||M_z||T_z)$ exactly as the honest sender would. It then samples $X_{1\oplus c} \in_{\mathrm{R}} \{0,1\}^{2\psi+1}$ and inputs (X_0, X_1) to B^*.

3. The corrupt receiver B^* replies by sending some $\overline{y} \in \{0,1\}$.

4. S sets $\overline{z} = r \oplus \overline{y}$, computes $I_{\overline{z}} = H(M_z) \oplus T_{1\oplus\overline{z}}$ and samples $I_{1\oplus\overline{z}} \in_{\mathrm{R}} \{0,1\}^\psi$. It then inputs (I_0, I_1) to B^*.

5. B^* outputs some $(\overline{T}_0, \overline{T}_1)$ for the $\mathcal{F}_{\mathrm{EQ}}$ functionality and S continues or aborts as the honest A would in the real protocol, depending on whether or not $(T_0, T_1) = (\overline{T}_0, \overline{T}_1)$.

For the analysis we denote by F the event that B^* queries the RO on $K_c \oplus (1 \oplus c)\Delta_B$ or $K_z \oplus (1 \oplus z)\Delta_B$. We first show that assuming F does not occur, the simulation is perfect. We then show that F only occurs with negligible probability (during the protocol) and thus the simulation is indistinguishable from the real protocol (during the protocol). We then discuss how to simulate the RO after outputs have been delivered.

First in the view of B^* step 1 of the simulation is clearly identical to the real protocol. Thus the first deviation from the real protocol appears in step 2 of the simulation where the $X_{1\oplus c}$ is chosen uniformly at random. However,

assuming F does not occur, B^* has no information on $H(K_c \oplus (1 \oplus c)\Delta_B)$ thus in the view of B^*, $X_{1\oplus c}$ in the real protocol is a one-time pad encryption of $(x_{1\oplus c}||M_{x_{1\oplus c}}||T_{x_{1\oplus c}})$. In other words, assuming F does not occur, to B^*, $X_{1\oplus c}$ is uniformly random in both the simulation and the real protocol, and thus all input to B^* up to step 2 is distributed identically in the two cases.

For steps 3 to 5 notice that in the real protocol an honest sender would set $K_z = K_r \oplus \overline{y}\Delta_B$ and we would have

$$(K_r \oplus \overline{y}\Delta_B) \oplus \overline{z}\Delta_B = K_r \oplus r\Delta_B = M_r \ .$$

Thus we have that the simulation generates $I_{\overline{z}}$ exactly as in the real protocol. An argument similar to the one above for step 2 then gives us that the simulation is perfect given the assumption that F does not occur.

We now show that B^* can be modified so that if F does occur, then B^* can find Δ_B. However, since all input to B^* are independent of Δ_B (during the protocol), B^* only has negligible probability of guessing Δ_B and thus we can conclude that F only occurs with negligible probability.

The modified B^* keeps a list $Q = (Q_1, \ldots, Q_q)$ of all B^*'s queries to H. Since B^* is efficient we have that q is a polynomial in ψ. To find Δ_B the modified B^* then goes over all $Q_k \in_{\mathrm{R}} Q$ and computes $Q_k \oplus M_z = \Delta'$ and $Q_k \oplus M_c = \Delta''$. Assuming that F does occur there will be some $Q_{k'} \in Q$ s.t. $\Delta' = \Delta_B$ or $\Delta'' = \Delta_B$. The simulator can therefore use global key queries to find Δ_B if F occurs.

We then have the issue that after outputs are delivered to the environment, the environment learns Δ_B, and we have to keep simulating H to the environment after outputs are delivered. This is handled exactly as in the proof of Theorem 4.3 using the programability of the RO. □

To deal with the leakage of the leaky $\mathcal{F}_{\mathrm{AOT}}$ functionality, we let B randomly partition and combine the leaky aOTs in buckets, in a way similar to how we dealt with leakage for $\mathcal{F}_{\mathrm{AAND}}$: the leaky AOTs in each bucket will be combined using an OT combiner (as shown in Figure 4.11), in so that if at least one choice bit in every bucket is unknown to A, then the resulting aOT will not be leaky. We prove the following theorem.

Theorem 4.6. *Let $\mathcal{F}_{\mathrm{AOT}}(\ell)$ denote the functionality that provides ℓ non-leaky aOTs as in $\mathcal{F}_{\mathrm{DEAL}}$. If $B \geq \frac{\sigma}{1+\log_2(\ell)} + 1$, then the protocol in Figure 4.11 securely implements $\mathcal{F}_{\mathrm{AOT}}(\ell)$ in the leaky $\mathcal{F}_{\mathrm{AOT}}(B\ell)$-hybrid model with statistical security parameter σ.*

Proof. We want to show that the protocol in Figure 4.11 provides non-leaky aOTs, having access to a functionality that provides leaky aOTs.

In the protocol the receiver randomly partitions ℓB leaky aOTs in ℓ buckets of size B. First we want to argue that the probability that every bucket contains at least one aOT where the choice bit is unknown to the adversary is overwhelming. Repeating the same calculations as in the proof of Theorem 4.4 it turns out that this happens with probability bigger than $1 - (2\ell)^{(1-B)} \geq 1 - 2^{-\sigma}$.

Once we know that (with overwhelming probability) at least one OT in every bucket is secure for the receiver (i.e., at least one choice bit is uniformly random

1. A and B generate $\ell' = B\ell$ authenticated OTs using the leaky $\mathcal{F}_{\text{AOT}}(\ell')$ functionality. If the functionality does not abort, name the outputs $\{[x_0^i]_{\mathsf{A}}, [x_1^i]_{\mathsf{A}}, [c^i]_{\mathsf{B}}, [z^i]_{\mathsf{B}}\}_{i=1}^{\ell'}$.
2. B sends a B-wise independent permutation π on $\{1, \ldots, \ell'\}$ to A. For $j = 0, \ldots, \ell - 1$, the B quadruples

$$\{[x_0^{\pi(i)}]_{\mathsf{A}}, [x_1^{\pi(i)}]_{\mathsf{A}}, [c^{\pi(i)}]_{\mathsf{B}}, [z^{\pi(i)}]_{\mathsf{B}}\}_{i=jB+1}^{jB+B}$$

   are defined to be in the j'th bucket.
3. A and B combine the B quadruples in each bucket. Here we describe how to combine two quadruples from in a bucket: let them be $[x_0^1]_{\mathsf{A}}, [x_1^1]_{\mathsf{A}}, [c^1]_{\mathsf{B}}, [z^1]_{\mathsf{B}}$ and $[x_0^2]_{\mathsf{A}}, [x_1^2]_{\mathsf{A}}, [c^2]_{\mathsf{B}}, [z^2]_{\mathsf{B}}$. Let the result be $[x_0]_{\mathsf{A}}, [x_1]_{\mathsf{A}}, [c]_{\mathsf{B}}, [z]_{\mathsf{B}}$. To combine more than two, just iterate by taking the result and combine it with the next quadruple in the bucket.
   (a) A reveals $d = x_0^1 \oplus x_1^1 \oplus x_0^2 \oplus x_1^2$.
   (b) A and B compute

$$[c]_{\mathsf{B}} = [c^1]_{\mathsf{B}} \oplus [c^2]_{\mathsf{B}} \ , [z]_{\mathsf{B}} = [z^1]_{\mathsf{B}} \oplus [z^2]_{\mathsf{B}} \oplus d[c^1]_{\mathsf{B}} \ ,$$

   and

$$[x_0]_{\mathsf{A}} = [x_0^1]_{\mathsf{A}} \oplus [x_0^2]_{\mathsf{A}} \ , [x_1]_{\mathsf{A}} = [x_0^1]_{\mathsf{A}} \oplus [x_1^2]_{\mathsf{A}} \ .$$

Figure 4.11: From Leaky Authenticated OTs to Authenticated OTs

in the view of the adversary), the security of the protocol follows from the fact that we use a standard OT combiner [HKN+05]. Turning this into a simulation proof can be easily done in a way similar to the proof of Theorem 4.4. □

This completes the description of our protocol.

4.5 Complexity Analysis

We will now sketch a complexity analysis counting the calls to symmetric primitives used in the TinyOT protocol.

As showed in Theorem 4.2, the protocol requires an initial call to an ideal functionality for $(\mathcal{F}_{\text{OT}}(\frac{44}{3}\psi, \psi), \mathcal{F}_{\text{EQ}}(\psi))$. After this, the cost per gate is only a number of invocations to a cryptographic hash function H. In this section we give the exact number of hash functions that we use in the construction of the different primitives. As the final protocol is completely symmetric, we count the total number of calls to H made by both parties.

Equality $\mathcal{F}_{\text{EQ}}$: The $\mathcal{F}_{\text{EQ}}$ functionality can be securely implemented with 2 calls to a hash function H, as described in Chapter 2

Authenticated OT $\mathcal{F}_{\text{AOT}}$: Every aOT costs $4B$ aBits, $2B$ calls to $\mathcal{F}_{\text{EQ}}$, and $6B$ calls to H, where B is the "bucket size".

Authenticated AND $\mathcal{F}_{\text{AAND}}$: Every aAND costs $3B$ aBits, B calls to $\mathcal{F}_{\text{EQ}}$, and $3B$ calls to H, where B is the "bucket size".

2PC Protocol, Input Gate: Input gates cost 1 aBit.

2PC Protocol, AND Gate: AND gates cost 2 aOTs, 2 aANDs and 2 aBits.

2PC Protocol, XOR Gate: XOR gates require no calls to H.

The cost per aBit, requires 59 calls to H as described in Chapter 3. However, using some further optimizations (that are not described in here, as they undermine the modularity of our constructions) we can take this number down to 8.

By plugging in these values we get that the cost per input gate is 59 calls to H (8 with optimizations), and the cost per AND gate is $856B + 118$ calls to H ($142B + 16$ with optimizations). The implementation described in Section 4.6 uses the optimized version of the protocol and buckets of fixed size 4, and therefore the total cost per AND gate is 584 calls to H.

As described in Section 4.1 we can greatly reduce communication complexity of our protocol by deferring the MAC checks. However, this trick comes at cost of two calls to H (one for each player) every time we do a "reveal". This adds $2B$ hashes for each aOT and aAND and in total adds $8B+20$ hashes to the cost each AND gate. This added cost is not affected by the optimization mentioned above.

4.6 Experimental Results

We did a proof-of-concept implementation in Java. The hash function in our protocol was implemented using Java's standard implementation of SHA256. The implementation consists of a circuit-independent protocol for preprocessing all the random values output by $\mathcal{F}_{\text{DEAL}}$, a framework for constructing circuits for a given computation, and a run-time system which takes preprocessed values, circuits and inputs and carry out the secure computation.

We will not dwell on the details of the implementation, except for one detail regarding the generation of the circuits. In our implementation, we do not compile the function to be evaluated into a circuit in a separate step. The reason is that this would involve storing a huge, often highly redundant, circuit on the disk, and reading it back. This heavy disk access turned out to constitute a significant part of the running time in an early prototype implementations which we discarded. Instead, in the current prototype, circuits are generated in chunks on the fly. The chunks size is balanced to be large enough that their evaluation generate large enough network packages that we can amortize away communication latency, but small enough that the circuit chunks can be kept in memory during their evaluation. A circuit compiled is hence replaced by a succinct program which generates the circuit in a streaming manner. This circuit stream is then sent through the runtime machine, which receives a separate stream of preprocessed $\mathcal{F}_{\text{DEAL}}$-values from the disk and then evaluates the circuit chunk by chunk in concert with the runtime machine at the other party in the protocol. The stream of preprocessed $\mathcal{F}_{\text{DEAL}}$-values from the disk is still expensive, but we currently see no way to avoid this disk access, as the random nature of the preprocessed values seems to rule out a succinct representation.

For timing we did oblivious ECB-AES encryption. (Both parties input a secret 128-bit key K_A respectively K_B, defining an AES key $K = K_A \oplus K_B$. A inputs a secret ℓ-block message $(m_1, \ldots, m_\ell) \in \{0,1\}^{128\ell}$. B learns $(E_K(m_1), \ldots, E_K(m_\ell))$.) We used the AES circuit from [PSSW09] and we thank Benny Pinkas, Thomas Schneider, Nigel P. Smart and Stephen C. Williams for providing us with this circuit.

The reason for using AES is that it provides a reasonable sized circuit which is also reasonably complex in terms of the structure of the circuit and the depth, as opposed to just running a lot of AND gates in parallel. Also, AES has been used for benchmark in previous implementations, like [PSSW09], which allows us to do a crude comparison to previous implementations. The comparison can only become crude, as the experiments were run in different experimental setups.

ℓ	G	σ	T_{pre}	T_{onl}	T_{tot}/ℓ	G/T_{tot}
1	34,520	55	38	4	44	822
27	922,056	55	38	5	1.6	21,545
54	1,842,728	58	79	6	1.6	21,623
81	2,765,400	60	126	10	1.7	20,405
108	3,721,208	61	170	12	1.7	20,541
135	4,642,880	62	210	15	1.7	20,637

ℓ	G	σ	T_{pre}	T_{onl}	T_{tot}/ℓ	G/T_{tot}
256	8,739,200	65	406	16	1.7	20,709
512	17,478,016	68	907	26	1.8	18,733
1,024	34,955,648	71	2,303	52	2.3	14,843
2,048	69,910,912	74	5,324	143	2.7	12,788
4,096	139,821,440	77	11,238	194	2.8	12,231
8,192	279,642,496	80	22,720	258	2.8	12,170
16,384	559,284,608	83	46,584	517	2.9	11,874

Figure 4.12: Timings. Left table is average over 5 runs. Right table is from single runs. Units are as follows: ℓ is number of 128-bit blocks encrypted, G is Boolean gates, σ is bits of security, $T_{\text{pre}}, T_{\text{onl}}, T_{\text{tot}}$ are seconds.

In the timings we ran A and B on two different machines on Aarhus University's intranet (using two Intel Xeon E3430 2.40GHz cores on each machine). We recorded the number of Boolean gates evaluated (G), the time spent in preprocessing (T_{pre}) and the time spent by the run-time system (T_{onl}). In the table in Figure 4.12 we also give the amortized time per AES encryption (T_{tot}/ℓ with $T_{\text{tot}} \stackrel{\text{def}}{=} T_{\text{pre}} + T_{\text{onl}}$) and the number of gates handled per second (G/T_{tot}). The time T_{pre} covers the time spent on computing and communicating during the generation of the values preprocessed by $\mathcal{F}_{\text{DEAL}}$, and the time spent storing these value to a local disk. The time T_{onl} covers the time spent on generating the circuit and the computation and communication involved in evaluating the circuit given the values preprocessed by $\mathcal{F}_{\text{DEAL}}$.

We work with two security parameters. The computational security parameter ψ specifies that a poly-time adversary should have probability at most $\text{poly}(\psi)2^{-\psi}$ in breaking the protocol. The statistical security parameter σ spec-

ifies that we allow the protocol to break with probability $2^{-\sigma}$ independent of the computational power of the adversary. As an example of the use of ψ, our keys and therefore MACs have length ψ. This is needed as the adversary learns $H(K_i)$ and $H(K_i \oplus \Delta)$ in our protocols and can break the protocol given Δ. As an example of the use of σ, when we generate ℓ gates with bucket size B, then $\sigma \leq (\log_2(\ell)+1)(B-1)$ due to the probability $(2\ell)^{1-B}$ that a bucket might end up containing only leaky components. This probability is independent of the computational power of the adversary, as the components are being bucketed by the honest party after it is determined which of them are leaky.

In the timings, the computational security parameter has been set to 120. Since our implementation has a fixed bucket size of 4, the statistical security level depends on ℓ. In the table, we specify the statistical security level attained (σ means insecurity $2^{-\sigma}$). At computational security level 120, the implementation needs to do 640 seed OTs. The timings do not include the time needed to do these, as that would depend on the implementation of the seed OTs, which is not the focus here. We note, however, that using, e.g., the implementation in [PSSW09], the seed OTs could be done in around 20 seconds, so they would not significantly affect the amortized times reported.

The dramatic drop in amortized time from $\ell = 1$ to $\ell = 27$ is due to the fact that the preprocessor, due to implementation choices, has a smallest unit of gates it can preprocess for. The largest number of AES circuits needing only one, two, three, four and five units is 27, 54, 81, 108 and 135, respectively. Hence we preprocess equally many gates when $\ell = 1$ and $\ell = 27$.

As for total time, we found the best amortized behavior at $\ell = 54$, where oblivious AES encryption of one block takes amortized 1.6 seconds, and we handle 21,623 gates per second. As for online time, we found the best amortized behavior at $\ell = 2048$, where handling one AES block online takes amortized 32 milliseconds, and online we handle 1,083,885 gates per second. We find these timings encouraging and we plan an implementation in a more machine-near language, exploiting some of the findings from implementing the prototype.

Part II

MiniLEGO

s

Chapter 5

Commitments

In this chapter we present our construction of XOR-homomorphic commitment based solely on OT. To the best of our knowledge this is the first kind of homomorphic cryptographic primitive that can been obtained under general assumption.

More precisely, we will implement a functionality for what we call *weak commitment with XOR-homomorphic opening*. By *weak* we mean that a corrupted sender is allowed to make a small amount of non-binding commitments which we call *wildcard commitments*. We allow A to open wildcard commitments to any value as long as the opening is consistent with all previous openings.

The idea behind our protocol is inspired by the watch lists of the constructions in [IKOS08, IPS08]. When A commits to a message m we let B learn t random bit positions (which we call the *watch bits*) of a linear error correcting encoding of m. To open the commitment A sends the message m to B, who encodes it and checks that the codeword matches the t bits he saw earlier. For A to break binding she would have to find a message $m' \neq m$ with a codeword that matches the codeword for m on B's random watch bits. If we use a code with a high minimum distance the probability that A finds such a message becomes very low.

Overview

- In Section 5.1 we give the ideal functionality for our commitments.
- In Section 5.2 we describe a generic error correcting code, with a certain privacy property we need for our implementation. Namely, we need that learning the t watch bits of a codeword does not help B to guess the encoded message.
- In Section 5.3 we describe our protocol: We use a $\binom{n}{t}$-$\mathcal{F}_{\text{ROT}}$ functionality to establish a one time pad key for for each commitment, in such a way that B learns t bits of the key. To commit A sends a one time pad encryption of the encoded message as described above. A problem turns out to be that A could add some errors to the encoding to help her cheat later. Therefore, we use the cut-n-choose technique to ensure the she

can at most do this for a few commitments (which becomes the wildcard commitments).

- We give the security analysis in Section 5.4. Here the main problem becomes to find the commitments to mark as wildcard commitments when A is corrupted. Doing this efficiently turns out to be essentially linked to showing that any errors A adds to the commitments, by the cut-n-choose test, must be isolated to a few common bit positions.

- Finally in Section 5.5 we give a few simple extensions to the base protocol that we will need for the MiniLEGO 2PC protocol in Chapter 6.

- We end the chapter in Section 5.6 by sketching a complexity analysis counting the symmetric primitives used in the protocol.

5.1 The Ideal Functionality

We name our ideal functionality $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ and describe it in Figure 5.1. The functionality allows A to commit to up to ℓ messages and to later reveal those messages. In addition the $\mathcal{F}_{\text{WCOM}}$ allows to reveal the XOR of two or more committed messages to B (without revealing any extra information about the original committed messages).

The functionality is "insecure", in the sense that a corrupt A can choose a set of up to K wildcard commitments where she can change her mind about the committed value at opening time. However, openings need to be consistent, e.g., if she opens the same commitment twice, she can freely choose which value to open to the first time, and then she has to stick to her choice the second time. More generally, every time A performs an opening this defines a linear equation on the K wildcards commitments, and these equations must be consistent.

We formalize this in the following way: First we let the adversary specify a set identifiers ID of size ℓ used to identify each of the ℓ commitments (for technical reasons ID will be a subset of $[2\ell]$). In addition the adversary gives a set $W \subset ID$, of size at most K, to identify the wildcard commitments. $\mathcal{F}_{\text{WCOM}}$ stores a set of linear equations on ℓ variables $(\mathtt{X}_i)_{i \in ID}$ (one for each commitment). Initially this set is empty. Each time A commits to a message m_j using a non-wildcard commitment (i.e. $j \in ID \setminus W$) $\mathcal{F}_{\text{WCOM}}$ will store the equation $\mathtt{X}_j = m_j$. For wildcard commitments no such equation are stored when A makes the commitment. If A instructs $\mathcal{F}_{\text{WCOM}}$ to open the XOR of the set of commitments $J \subset ID$ we let corrupted A input a message $m_J \in \{0,1\}^\psi$ she wants to open to. The functionality then adds the equation $\bigoplus_{j \in J} \mathtt{X}_j = m_J$ to the set of stored equations and checks that this set of equations has a solution, i.e., if there is an assignment of values in $\{0,1\}^\psi$ to each variable $\mathtt{X}_j$ such that all equations are satisfied. If so, the functionality permanently stores the equation $\bigoplus_{j \in J} \mathtt{X}_j = m_J$ and opens the XOR of the set of commitments J as m_J. Otherwise, $\mathcal{F}_{\text{WCOM}}$ will output `Alice cheats` to B and terminate. Note that if $J \cap W = \emptyset$ then for all $j \in J$ the functionality has stored the equation $\mathtt{X}_j = m_j$, and therefore a corrupt A can only open the commitment successfully

if $m_J = \bigoplus_{j \in J} m_j$. Note also that if, e.g., A has made commitments $i \in W$ and $j \in ID \setminus W$, and opens the XOR of commitments i and j as m' then for all later openings A can only successfully open the wildcard commitment i as $m'_i = m_j \oplus m'$. In these cases, when a commitment can only successfully be opened to one value, we say that the commitment is *fixed* to that value. Non-wildcard commitments are always fixed and wildcard commitments can become fixed when opened XOR a non-wildcard commitment. When a wildcard commitment has been fixed it can essentially be viewed as a non-wildcard commitment.

Notice that the terminology can become a little confusing because of the wildcard commitments: when we say that A opens the XOR of some set of commitments $J \subset ID$ to a value m_J, then we can not guarantee that $m_J = \bigoplus_{j \in J} m_j$, when $J \cap W \neq \emptyset$.

Init

On input (ID, W) with $|ID| = \ell$, $|W| \leq K$ and $W \subset ID$ from the adversary output ID to both parties and let $\mathcal{J} = \emptyset$. If A is honest, then $W = \emptyset$.

Commit

On input $(\texttt{commit}, j \in ID, m_j)$ with $m_j \in \{0,1\}^\psi$ from A, and where no value of the form $(j, \cdot)$ is stored, store (j, m_j). If $j \in ID \setminus W$, add $J = \{j\}$ to $\mathcal{J}$ and associate with J the equation $\mathtt{X}_j = m_j$. Then output $(\texttt{commit}, j)$ to B.

Open

On input $(\texttt{open}, J \subset ID)$ from A, where for all $j \in J$ a pair (j, m_j) is stored do the following:

- If A is honest, output $(\texttt{open}, J, \oplus_{j \in J} m_j)$ to B.
- If A is corrupted wait for A to input $(\texttt{corrupt-open}, J, m_J)$. Then add J to $\mathcal{J}$, associate the equation $\oplus_{j \in J} \mathtt{X}_j = m_J$ to J, and check that the equation system $\{\oplus_{j \in J} \mathtt{X}_j = m_J\}_{J \in \mathcal{J}}$ has a solution[a]. If so, output $(\texttt{open}, J, m_J)$ to B. Otherwise, output `Alice cheats` to B and terminate.

[a] I.e., there should be an assignment of values to the wildcard commitments such that all stored openings can be explained by this assignment.

Figure 5.1: The ideal functionality $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ for ℓ weak commitments.

5.2 Error Correcting Code

In the implementation of the functionality described above we need an error correcting code (ECC), which encodes an ψ-bit string as an n-bit string with minimal distance at least d using some ρ-bits of randomness. It should at the same time be a secret sharing scheme in that seeing t random positions of a random codeword does not leak information on the message. We denote this scheme by $\text{ssecc}^{\psi,n,d,t}$. We use $\text{enc}^{\psi,n,d,t}$ to denote the encoding function and we use $\text{dec}^{\psi,n,d,t}$ to denote the decoding function. Both should be PPT and we drop parameters for notational convenience. The code should have the following properties.

Error correction For all $m \in \{0,1\}^\psi$, $r \in \{0,1\}^\rho$ and error vectors $e \in \{0,1\}^n$ with $\text{hw}(e) < d/2$ it holds that $\text{dec}(\text{enc}(m; r) \oplus e) = m$, where

hw is the Hamming weight in $\{0,1\}^n$. We assume that $\mathrm{dec}(C) = \perp$ when C has distance more than $d/2$ to all codewords and we assume that there exists an efficient algorithm ncw (nearest codeword) such that $\mathrm{ncw}(\mathrm{enc}(m;r) \oplus e) = \mathrm{enc}(m;r)$ when $\mathrm{hw}(e) < d/2$.

Privacy There exists a PPT function xpl which can explain any codeword as being a codeword of any message to anyone who knows at most t positions of the codeword. Formally, for all $I \subset [n]$, $|I| = t$ and all $m, m' \in \{0,1\}^\psi$ the distributions D_0 and D_1 described below are statistically close. The distribution D_0 is generated as follows: sample $r \in_{\mathrm{R}} \{0,1\}^\rho$, let $c = \mathrm{enc}(m;r)$ and output $((c_i)_{i\in I}, m, r)$. The distribution D_1 is generated as follows: sample $r' \in_{\mathrm{R}} \{0,1\}^\rho$, let $c = \mathrm{enc}(m';r')$, sample $r \leftarrow \mathrm{xpl}(I, m', r', m)$ and output $((c_i)_{i\in I}, m, r)$.

Linearity For all $m, m' \in \{0,1\}^\psi$ and $r, r' \in \{0,1\}^\rho$ it holds that $\mathrm{enc}(m;r) \oplus \mathrm{enc}(m';r') = \mathrm{enc}(m \oplus m'; r \oplus r')$.

Note that **Error correction** implies that the minimal distance is at least d, i.e., for all $m \neq m' \in \{0,1\}^\psi$ and $r, r' \in \{0,1\}^\rho$, $c = \mathrm{enc}(m;r)$ and $c' = \mathrm{enc}(m';r')$ it holds that $\mathrm{ham}(c,c') \geq d$ where ham is the Hamming distance.

We further require of the parameters of ssecc that: $n = \Theta(\sigma)$, $t = \Theta(n)$, $d = \Theta(n)$. I.e. both the privacy and minimum distance of ssecc must be a constant fraction of the length of codewords.

Codes that satisfy the desired properties can be found in [CC06].

5.3 Protocol

The protocol π_{WCOM} implementing $\mathcal{F}_{\mathrm{WCOM}}$ is described formally in Figure 5.2. Here we describe the ideas behind it.

Let $v \in \{0,1\}^n$ and $I = \{i_1, i_2, \ldots, i_t\} \subseteq [n]$. We define the function $w_I : \{0,1\}^n \to \{0,1\}^t$ so that $w_I(v) = (v_{i_1}, v_{i_2}, \ldots, v_{i_t}) \in \{0,1\}^t$, i.e., $w_I(v)$ is the t-bit string consisting of the t bits in v indexed by I.

In the protocol a commitment to a message m is a one-time pad of m with some key T. Clearly this is hiding but not binding. To make the commitment binding we allow the receiver of the commitment (B) to learn $w_I(m)$ for some secret set $I \subseteq [n]$. We denote $w_I(m)$ the *watch bits* of the commitment. To open the commitment to m A sends m' to B and B checks if $w_I(m') = w_I(m)$. If this is not the case B rejects the opening.

The watch bits gives some degree of binding since A can only open the commitment to some message $m' \neq m$ if $w_I(m') = w_I(m)$. I.e. if $t = |I|$ is large enough A can only hope to change a few bits of m without getting caught. On the other hand the watch bits clearly compromises the hiding property of the commitment. To avoid this we use the code ssecc to encode the message m and commit to the encoded m instead. I.e. a commitment to m becomes $\mathrm{enc}(m;r) \oplus T$ where. By privacy of ssecc m is now hidden.

The encoding additionally strengthens binding of the commitment: codewords c and c' encoding to two different messages m and m' must be different

in many bit positions. Thus for A to open a commitment to m to m' none of these positions must be in the watch bits.

More precisely let $d = 2s + 1$ be the minimum distance of ssecc for some $s < \frac{n}{2}$. Suppose a corrupt A gives the commitment, $c \oplus T$. Note that when A is corrupt c does not have to be a codeword. In that case we have that $c = \mathrm{ncw}(c) \oplus e$ for some *error vector* $e \in \{0,1\}^n$, and we say the commitment has $\mathrm{hw}(e)$ *errors*.

Regardless of the number of errors, consider what it takes for A to be able to open this commitment to two different messages m' and m'', with codewords c' and c'' respectively: for any two different codewords c' and c'' one of them has distance at least s to c, say c'. In other words c' has at least s bit positions different from c. If A tries to open the commitment to m', B only accepts the opening if none of these bit positions are in his t watch bits for the commitment. Thus for any commitment (possibly with errors) the probability that a cheating A can open the commitment to two different messages m' and m'' is at most

$$\begin{aligned}\binom{n-t}{s}\binom{n}{s}^{-1} &= \prod_{i=0}^{s-1}\frac{n-t-i}{s-i}\prod_{i=0}^{s-1}\frac{s-i}{n-i} \\ &= \prod_{i=0}^{s-1}\frac{n-t-i}{n-i} \\ &= \prod_{i=0}^{s-1}1-\frac{t}{n-i}\ .\end{aligned}$$

Assume that $s = s'n$ and $t = t'n$ for constants $0 < t', s' < 1$, then the probability of A breaking the binding property is at most

$$\begin{aligned}\prod_{i=0}^{s-1}\left(1-\frac{t}{n-i}\right) &\leq \prod_{i=0}^{s-1}\left(1-\frac{t'n}{n}\right) \\ &= \left(1-t'\right)^{s'n}\ .\end{aligned}$$

Thus with σ being the security parameter, $n = \Theta(\sigma)$ and for any positive constants t', s' with $0 < t', s' < 1$ A will have negligible probability of breaking binding.

Notice, that while c' will have distance at least s to c it could be that c'' is much closer to c. E.g. c'' could be the nearest codeword to c. In this case, by a similar calculation as the one above, we have that, if the commitment has *very few* errors, a cheating A *could* open the commitment to m'' with noticeable probability (say, if the number of errors were constant in σ). This is not a problem since such a "slightly wrong" commitment can be seen as a commitment to the message m'' encoded by the nearest codeword c''.

To get XOR-homomorphic commitments, more work has to be done. The problem being that the XOR of several commitments with errors, may become a commitment that breaks binding, even if the individual commitments only have a few errors. Consider a number of commitments made with non-codewords c_i with nearest codewords c_i'. The XOR the non-codewords $c = \bigoplus_i c_i$ may then

be very far from the XOR of their nearest codewords $c' = \bigoplus_i c'_i$. In fact c might be so far away from c' that it gets very close to some other codeword c''. Hence the XOR of the commitments can be opened to a message different from the XOR of the message associated with the individual commitments. This would break the binding property.

To deal with this problem, the protocol π_{WCOM} starts by letting A commit to 2ℓ random messages. We then do a cut-n-choose test to check that half of these commitments can be opened correctly. If A passes the test we have that, with overwhelming probability, the remaining commitments only have a few errors. Additionally, those errors must be isolated to a few common bit positions. Thus the result of XOR'ing these commitments will at most have a few a errors in the same positions.

Thus if A passes the cut-n-choose test we use the un-tested random commitments to implement the actual commitments. The resulting commitment will have exactly the same errors as the random commitment (if any).

The details of the protocol π_{WCOM} is given in Figure 5.2. In Section 5.4 we prove the following theorem.

Theorem 5.1. *Let σ be the security parameter, $K = n\sigma + \sigma$ and let* ssecc *be a code with $n = \Theta(\sigma)$, $t = \Theta(n)$, $d = \Theta(n)$ and $\sigma < d/2$ as, e.g., given by [CC06]. Then the protocol in Figure 5.2 UC, active, static securely implements $\mathcal{F}_{\mathrm{WCOM}}^{K,\ell}$ in the $\binom{n}{t}$-$\mathcal{F}_{\mathrm{ROT}}(2\ell)$-hybrid model.*

5.4 Analysis

Simulating when no party is corrupted or both parties are corrupted is straight forward. Simulating when B is corrupted is also quite simple, and can be done using standard techniques from simulation in secure multiparty computation based on secret sharing. Thus below we will only sketch the proof for corrupted B, and focus the case of corrupted A.

5.4.1 Corrupted B

The simulator commits to 0^ψ in all commitments. When asked to open such a commitment U_j to a given $m_j \in \{0,1\}^\psi$ it uses the efficient algorithm xpl to explain the commitment as $U_j = \mathrm{enc}(x_j; r_j) \oplus T_j$ for $x_j = y_j \oplus m_j$. The only non-trivial detail is that if the simulator is asked to open a commitment, where the value of the opening follows from previous openings (i.e., using some linear equation), it computes the opening as a linear combination of the previous simulated openings. As an example, if the simulator opened U_j as $U_j = \mathrm{enc}(x_j; r_j) \oplus T_j$ and opened U_i as $U_i = \mathrm{enc}(y_i; r_i) \oplus T_i$. Then it will open $U_j \oplus U_i$ as $U_j = \mathrm{enc}(x_j \oplus x_i; r_j \oplus r_i) \oplus T_j \oplus T_i$.

5.4.2 Corrupted A

Intuition of the proof when A is corrupted is that the cut-n-choose test will catch A if there are many indices i for which there exists a commitment that

Setup

To set up the scheme A and B run the following.

1. A and B run a $\binom{n}{t}$-$\mathcal{F}_{\mathrm{ROT}}(2\ell)$ functionality and get as output $(R_i)_{i\in[n]}$ and $(I,(R_i)_{i\in I})$ respectively where $R_1,\ldots,R_n \in_{\mathrm{R}} \{0,1\}^{2\ell}$ and I is a uniformly random subset of $[n]$ with $|I|=t$.
2. A lets $R \in \{0,1\}^{n\times 2\ell}$ be the matrix with R_i as the i'th row and lets $T_j \in \{0,1\}^n$ be the j'th column of R. [a]
3. A for $j=1,\ldots,2\ell$, samples $x_j \in_{\mathrm{R}} \{0,1\}^{\psi}$, $r_j \in_{\mathrm{R}} \{0,1\}^{\rho}$ and sends the commitment $c_j = (\mathrm{enc}(x_j;r_j)\oplus T_j, j)$. Let $c_j=(U_j,j)$ the value received by B.
4. B sends a uniformly random subset $C\subset[2\ell]$. This also defines $ID=\bar{C}$.
5. For $j\in C$, A opens c_j by sending $o_j=(x_j,r_j,j)$.
6. For $j\in C$, B receives (x'_j,r'_j,j) and checks that

$$w_I(\mathrm{enc}(x'_j;r'_j)) = w_I(U_j)\oplus w_I(T_j)\ ,$$

if not B terminates the protocol.

Commit

To commit to m_j for $j\in ID$ A sends (y_j,j) to B where y_j is the *correction value* $y_j = x_j\oplus m_j$.

Open

To open the XOR of commitments $J\subset ID$ the parties do the following.

1. For $j\in J$, let $c_j=(\mathrm{enc}(x_j;r_j)\oplus T_j,j)$ be the commitments sent in initialization and y_j the value sent during commitment. A computes the opening of $\bigoplus_{j\in J} m_j$ as $o_J=(\bigoplus_{j\in J}x_j,\bigoplus_{j\in J}r_j,J)$, and sends it to B.
2. If an opening of J was done previously, B uses the previous m_J, otherwise he proceeds as follows: Let $c_j=(U_j,j)$ be the commitments received during **Setup**. B accepts $o_J=(x_J,r_J,J)$ iff

$$w_I(\mathrm{enc}(x_J;r_J)) = w_I(\bigoplus_{j\in J}U_j)\oplus w_I(\bigoplus_{j\in J}T_j)\ ,$$

where

$$w_I(\bigoplus_{j\in J}U_j) = \bigoplus_{j\in J} w_I(U_j)$$

and

$$w_I(\bigoplus_{j\in J}T_j) = \bigoplus_{j\in J} w_I(T_j)\ .$$

If B accepts he outputs $x_J\oplus y_J$, where $y_J=\bigoplus_{j\in J}y_j$. Otherwise, B rejects the opening and terminates the protocol.

[a] Notice B can use $(R_i)_{i\in I}$ to compute $w_I(T_j)$ for all $j\in[2\ell]$.

Figure 5.2: The protocol π_{WCOM} implementing $\mathcal{F}_{\mathrm{WCOM}}$

has an error in position i. This is because if the errors of the commitments are very spread out, with high probability, many of them will be in the watch bits positions. As mentioned above, this means that almost all errors must be isolated in a few positions. Therefore XOR's of commitments will also have errors only in these position, and therefore will also be close to their "correct" codeword. The proof is complicated by the fact that a few commitments with

many errors, or errors outside the few isolated positions, may pass the cut-n-choose test. These commitments will be the wildcards. It can be shown that not even a commitment with many errors can be opened to two different values, as it would give a codeword encoding a non-zero value which is 0 in all the watch bits, which happens with negligible probability by the watch bits being random and the minimal distance high. This translates into it being impossible to make any combination of openings of linear equations yielding inconsistent outputs.

Proof. We first describe how to simulate the setup phase against a corrupt A^*. The simulator simulates the ideal functionality $\binom{n}{t}$-$\mathcal{F}_{\mathrm{ROT}}(\ell)$ to the adversarial A^* and records the outputs $(R_i)_{i\in[n]}$ and $(I,(R_i)_{i\in I})$. The simulator lets $R \in \{0,1\}^{n\times 2\ell}$ be the matrix with R_i as the i'th row and lets $T_j \in \{0,1\}^n$ be the j'th column of R. Then the simulator runs the rest of the setup as B would have done. What remains is to input to the ideal functionality $ID = \bar{C}$ along with a set W specifying which commitments should be wildcard commitments.

The simulator computes W as follows: When A^* sends a commitment (U_j, j), compute $S_j = U_j \oplus T_j$, let $N_j = \mathrm{ncw}(S_j)$ and let $e_j = N_j \oplus S_j$, where ncw gives the nearest codeword, i.e., e_j is the smallest error of S_j relative to the error correcting code. Let $E \in \{0,1\}^{n\times 2\ell}$ be the matrix with e_i as the i'th column. [1] In Lemma 5.1 and 5.2 we show that, if A^* passes the cut-n-choose test, with overwhelming probability there exists sets $F \in [2\ell]$ and $D \in [n]$ with $|D|, |F| \leq \sigma$, so that for all $E_{i,j} = 0$ if and only if $(i,j) \in D \times F$. In other words all commitments with in $[2\ell] \setminus F$ has all of their errors isolated to indices in D. In Lemma 5.3 we show that, if such sets exist then the simulator can efficiently compute sets D and F with the same properties except $|F| \leq n\sigma + \sigma = O(\sigma^2)$. Thus the simulator will compute these sets and input $(ID, W) = (\bar{C}, F \cap \bar{C})$ to $\mathcal{F}_{\mathrm{WCOM}}$ to end simulation of the setup. The simulation of the setup phase is perfect.

To simulate a commitment, record the received value (y_j, j). Let $x_j = \mathrm{dec}(N_j)$, for the value N_j computed when the j'th commitment was received in the setup phase as described above. Let $m_j = y_j \oplus x_j$, and input $(\texttt{commit}, j, m_j)$ to the ideal functionality. Again the simulation is clearly perfect.

To simulate an opening let $c_j = (U_j, j)$ be the commitments sent earlier and let y_j be the correction value sent at commit time. Let $o_J = (x_J, r_J, J)$ be the opening. Iff

$$w_I(\mathrm{enc}(x_J; r_J)) = w_I(\bigoplus_{j\in J} U_j) \oplus w_I(\bigoplus_{j\in J} T_j) \ ,$$

input $(\texttt{open}, J)$ to the ideal functionality—when J was not previously opened and $W \cap J \neq \emptyset$ follow this input by $(\texttt{corrupt-open}, J, m_J)$, where $m_J = x_J \oplus$

[1] Above and in the following we assume that ncw gives the closest codeword on all inputs, also those with distance more than $d/2$ to all codewords. Note that the simulator cannot necessarily compute $N_j = \mathrm{ncw}(S_j)$ efficiently when there are more than $d/2$ errors in S_j (i.e., when $\mathrm{hw}(e_j) > d/2$). However, from $\sigma < d/2$, it follows that $N_j = \mathrm{ncw}(S_j)$ is correct when $\mathrm{dec}(S_j) \neq \perp$. So, if the simulator lets $e_j = 1^n$ when $\mathrm{dec}(S_j) = \perp$, it will only add too many 1 to a column e_j where there would be more than σ non-zero entries anyway. It can be seen that this maintains the correctness of the below analysis, but for notational convenience we simply assume that ncw works for all inputs.

y_J and $y_J = \bigoplus_{j \in J} y_j$. If $w_I(\mathrm{enc}(x_J; r_J)) \neq w_I(\bigoplus_{j \in J} U_j) \oplus w_I(\bigoplus_{j \in J} T_j)$ the simulator rejects the opening and terminates the protocol.

It is clear that this simulation is perfect until one of the following two events occur:

1. During the simulation of an opening, the simulator inputs $(\texttt{open}, J)$ to the ideal functionality and $J \cap W = \emptyset$ and it is not the case that $m_J = \bigoplus_{j \in J} m_j$ for the m_j stored in the ideal functionality and the m_J computed by the simulator.

2. During the simulation of an opening, the simulator inputs $(\texttt{open}, J)$ to the ideal functionality followed by $(\texttt{corrupt-open}, J, m_J)$ and the equation $m_J = \bigoplus_{j \in J} m_j$ is not consistent with the equations stored in $\mathcal{F}_{\mathrm{WCOM}}$.

We prove that each of these events occur with negligible probability.

We start with the first case. For $j \in J$, let m_j be the values computed by the simulator, i.e.,

$$S_j = U_j \oplus T_j \ , \quad N_j = \mathrm{ncw}(S_j) \ , \quad e_j = N_j \oplus S_j \ , \quad x_j = \mathrm{dec}(N_j) \ , \quad m_j = y_j \oplus x_j \ .$$

For the first event to happen we have that A^* opened the commitment to some m_J for which

$$m_J \neq \bigoplus_{j \in J} m_j \ ,$$

since

$$\bigoplus_{j \in J} m_j = y_J \oplus \bigoplus_{j \in J} x_j \ .$$

Since the opening was of the form $o_J = (x_J, r_J, J)$ with

$$w_I(\mathrm{enc}(x_J; r_J)) = w_I(\bigoplus_{j \in J} U_j) \oplus w_I(\bigoplus_{j \in J} T_j) \tag{5.1}$$

and the output of the opening is defined to be

$$m_J = y_J \oplus x_J \ ,$$

we get that

$$x_J \neq \bigoplus_{j \in J} x_j \ , \tag{5.2}$$

which we will take as the basis for our contradiction. Note for later use how the correction values y_j cancelled out and $m_J \neq \bigoplus_{j \in J} m_j$ became $x_J \neq \bigoplus_{j \in J} x_j$. We later use this to simplify the proof of the more involved second event.

From $S_j = U_j \oplus T_j$ and (5.1) we get

$$w_I(\mathrm{enc}(x_J; r_J)) = w_I(\bigoplus_{j \in J} S_j) \ .$$

Using the argument from the discussion in Section 5.3, it follows from the above equation that except with negligible probability

$$\mathrm{dec}(\bigoplus_{j \in J} S_j) = x_J \ . \tag{5.3}$$

Namely, if $\mathrm{dec}(\bigoplus_{j\in J} S_j) \neq x_J$, then $\bigoplus_{j\in J} S_j$ and $\mathrm{enc}(x_J; r_J)$ would have Hamming distance at least $d/2$ and then $w_I(\mathrm{enc}(x_J; r_J)) = w_I(\bigoplus_{j\in J} S_j)$ occurs with negligible probability.

It follows from the way we pick $W = F \cap ID$ (as described above) and from $j \notin W$ for $j \in J$, that $\mathrm{hw}(e_j) \leq \sigma$ for all $j \in J$. I.e. e_j only has non-zero entries with indices in D and $|D| \leq \sigma$. Furthermore, from this it also follows that for all $j \in J$ the at most σ errors in each e_j are sitting in the same σ positions (those indexed by D). I.e.,

$$\mathrm{hw}(\bigoplus_{j\in J} e_j) \leq \sigma \ .$$

From $S_j = N_j \oplus e_j$ we get that

$$\bigoplus_{j\in J} S_j = \left(\bigoplus_{j\in J} N_j\right) \oplus \left(\bigoplus_{j\in J} e_j\right) \ ,$$

i.e.,

$$\mathrm{ham}(\bigoplus_{j\in J} S_j, \bigoplus_{j\in J} N_j) \leq \sigma \ ,$$

so

$$\mathrm{ncw}(\bigoplus_{j\in J} S_j) = \bigoplus_{j\in J} N_j \ .$$

Since

$$\mathrm{dec}(\bigoplus_{j\in J} N_j) = \bigoplus_{j\in J} \mathrm{dec}(N_j) = \bigoplus_{j\in J} x_j \ ,$$

we get that

$$\mathrm{dec}(\bigoplus_{j\in J} S_j) = \bigoplus_{j\in J} x_j \ . \tag{5.4}$$

Equations (5.2), (5.3) and (5.4) are in contradiction. Thus the first event occurs with at most negligible probability.

We then handle the second event. For simplicity we will assume that all $y_j = 0^\psi$ such that $m_j = x_j$ and $m_J = x_J$. This is without loss of generality qua the comment made above about the y_j-values cancelling out.

Note that all equations stored in $\mathcal{F}_{\mathrm{WCOM}}$ are of the form

$$\bigoplus_{j\in J'} \mathtt{X}_j = x_{J'} \ , \tag{5.5}$$

for some constant value $x_{J'}$ and some $J' \subset ID$. For any J' we can rewrite (5.5) stored in $\mathcal{F}_{\mathrm{WCOM}}$ as

$$\bigoplus_{j\in J'\cap W} \mathtt{X}_j = \left(\bigoplus_{j\in J'\setminus W} \mathtt{X}_j\right) \oplus x_{J'} \ .$$

If we let

$$\xi_{J'} = \left(\bigoplus_{j\in J'\setminus W} x_j\right) \oplus x_{J'} \ ,$$

we can then rewrite (5.5) as an equation

$$\bigoplus_{j \in J' \cap W} \mathtt{X}_j = \xi_{J'} \ ,$$

where $\xi_{J'}$ is fixed by the stored (x_j, j) for $j \notin W$ and the constants $x_{J'}$.

By assumption the system $\{\bigoplus_{j \in J' \cap W} \mathtt{X}_j = \xi_{J'}\}_{J' \in \mathcal{J}}$ is inconsistent. It is an easy piece of linear algebra to see that this means that there exists $\mathcal{J}_1, \mathcal{J}_2 \subseteq \mathcal{J}$ and $V \subset W$ such that

$$\bigoplus_{j \in V} \mathtt{X}_j = \bigoplus_{J' \in \mathcal{J}_1} \left(\bigoplus_{j \in J' \cap W} \mathtt{X}_j \right)$$

$$\bigoplus_{j \in V} \mathtt{X}_j = \bigoplus_{J' \in \mathcal{J}_2} \left(\bigoplus_{j \in J' \cap W} \mathtt{X}_j \right)$$

$$\bigoplus_{J' \in \mathcal{J}_1} \xi_{J'} \neq \bigoplus_{J' \in \mathcal{J}_2} \xi_{J'} \ .$$

(Simply apply Gaussian elimination to solve the system. This must fail, which will yield a system as above.)

From the last equation we get that

$$\bigoplus_{J' \in \mathcal{J}_1} \left(\left(\bigoplus_{j \in J' \setminus W} x_j \right) \oplus x_{J'} \right) \neq \bigoplus_{J' \in \mathcal{J}_2} \left(\left(\bigoplus_{j \in J' \setminus W} x_j \right) \oplus x_{J'} \right) \ ,$$

which implies that

$$\bigoplus_{J' \in \mathcal{J}_1 \cup \mathcal{J}_2} \left(\bigoplus_{j \in J' \setminus W} x_j \right) \neq \bigoplus_{J' \in \mathcal{J}_1 \cup \mathcal{J}_2} x_{J'} \ . \tag{5.6}$$

Note that the existence of the equation

$$\bigoplus_{j \in J' \cap W} \mathtt{X}_j = \left(\bigoplus_{j \in J' \setminus W} x_j \right) \oplus x_{J'} \ ,$$

stored in $\mathcal{F}_{\text{WCOM}}$, implies that A^* sent a codeword $S_{J'} = \text{enc}(x_{J'}; r_{J'})$ such that

$$w_I(S_{J'}) = w_I(\bigoplus_{j \in J'} S_j) \ ,$$

which means that

$$w_I(\bigoplus_{j \in J' \cap W} S_j) = w_I(\bigoplus_{j \in J' \setminus W} S_j) \oplus w_I(S_{J'}) \ .$$

Combining, we get that

$$w_I(\bigoplus_{j \in V} S_j) = \bigoplus_{J' \in \mathcal{J}_1} w_I(\bigoplus_{j \in J' \setminus W} S_j) \oplus w_I(S_{J'})$$

$$w_I(\bigoplus_{j\in V} S_j) = \bigoplus_{J'\in\mathcal{J}_2} w_I(\bigoplus_{j\in J'\setminus W} S_j) \oplus w_I(S_{J'}) \ .$$

Using transitivity of $=$ and that $w_I(S_{J'}) = w_I(\mathrm{enc}(x_{J'}; r_{J'}))$ we conclude that

$$\bigoplus_{J'\in\mathcal{J}_1} \left(w_I(\bigoplus_{j\in J'\setminus W} S_j) \oplus w_I(\mathrm{enc}(x_J; r_{J'})) \right) =$$
$$\bigoplus_{J'\in\mathcal{J}_2} \left(w_I(\bigoplus_{j\in J'\setminus W} S_j) \oplus w_I(\mathrm{enc}(x_{J'}; r_{J'})) \right) \ ,$$

which, implies that

$$w_I(\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \left(\bigoplus_{j\in J'\setminus W} S_j \right)) = w_I(\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \mathrm{enc}(x_{J'}; r_{J'})) \ . \tag{5.7}$$

By construction of W, the string $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \left(\bigoplus_{j\in J'\setminus W} S_j\right)$ has distance at most σ to a codeword encoding $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \left(\bigoplus_{j\in J'\setminus W} x_j\right)$, namely, they differ in at most the σ positions in the set D, as we sum only over $j \notin W$. By linearity, the string $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \mathrm{enc}(x_{J'}; r_{J'})$ is a codeword, encoding $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} x_{J'}$. By (5.7) and I being chosen at random and independent of the view of A^*, we have that

$$\mathrm{ham}(\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \left(\bigoplus_{j\in J'\setminus W} S_j \right), \bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \mathrm{enc}(x_{J'}; r_{J'})) \leq \sigma$$

except with negligible probability. It then follows from $\sigma < d/2$ that with overwhelming probability the string $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \mathrm{enc}(x_{J'}; r_{J'})$ is the only codeword within distance σ of $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \left(\bigoplus_{j\in J'\setminus W} S_j\right)$. Hence $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \mathrm{enc}(x_{J'}; r_{J'})$ must be the codeword encoding $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \left(\bigoplus_{j\in J'\setminus W} x_j\right)$. From this it follows that $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \mathrm{enc}(x_{J'}; r_{J'})$ is an encoding of both $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} x_{J'}$ and $\bigoplus_{J'\in\mathcal{J}_1\cup\mathcal{J}_2} \left(\bigoplus_{j\in J'\setminus W} x_j\right)$. This contradicts (5.6), and thus the second event can only occur with negligible probability. This concludes the proof. □

Now all that remains is to show that the simulator can indeed pick a set W as described in the proof. I.e. that we can pick sets $F \subset [2\ell]$ and $D \subset [n]$, where $|D| \leq \sigma$ and $|F| \leq O(\sigma^2)$, so that all commitments in $[2\ell] \setminus F$ have their errors isolated to indices in D.

To this end let a set of *independent errors*, be a set $\{(r_i, c_i)\}_{i=1}^B$ for which $r_i \in [n]$, $c_i \in [2\ell]$, $|\{r_i\}_{i=1}^B| = |\{c_i\}_{i=1}^B| = B$ and $E_{r_i,c_i} = 1$ for $i = [B]$ (recall that E is the matrix with the error vectors of all commitments as its columns). Let the *independent errors value* $\nu(E)$ to be the size of the largest set of independent errors.

Lemma 5.1. *The probability that A^* is not caught cheating during the setup is $\leq \alpha^{\nu(E)}$ for a positive constant $\alpha < 1$ independent of ν. In particular, if $\nu(E) = \Theta(\sigma)$, then A^* is caught except with overwhelming probability.*

Proof. To see this, let $\{(r_i, c_i)\}_{i=1}^{\nu}(E)$ be a maximal set of independent errors. Note that if $j \in C$ for some $j = c_i \in \{c_i\}_{i=1}^{\nu}(E)$ for the challenge C sent by B, i.e. A^* is asked to open commitment number $j = c_i$, she must send an opening (x'_j, r'_j, j) with $\mathrm{enc}(x'_j; r'_j) = N_j$. Assume namely that she does not: Then $\mathrm{ham}(\mathrm{enc}(x'_j; r'_j), S_j) \geq d/2$, and from the above it also follows that $w_I(U_j) \oplus w_I(T_j) = w_I(U_j \oplus T_j) = w_I(S_j)$, so $w_I(S_j) = w_I(\mathrm{enc}(x'_j; r'_j))$. Since I is independent of the view of A^*, the probability that $w_I(A) = w_I(B)$, when $\mathrm{ham}(A, B) \geq d/2$ is at most $(\frac{n-d/2}{n})^t$, which is negligible. If A^* does use an opening (x'_j, r'_j, j) with $\mathrm{enc}(x'_j; r'_j) = N_j$, then $e_j = \mathrm{enc}(x'_j; r'_j) \oplus S_j$ has a 1 in position r_i, so A^* is caught with probability at least $\frac{t}{n}$. The probability that $j = c_i$ is chosen for opening, i.e., $j \in C$, is at least $\frac{1}{2}$. This means the probability A^* is not caught is at most $\alpha = \frac{1}{2}(1 - \frac{n-t}{n})$. This holds independently for the $\nu(E)$ errors (r_i, c_i) as they sit in different rows and columns by the definition of ν. □

By Lemma 5.1 we can henceforth without loss of generality assume that the independent error value of E is less than σ.

We then let a pair of *error isolating* sets be two sets $D \subset [n]$ and $F \subset [2\ell]$ such that if $i \in [n] \setminus D$ and $j \in [2\ell] \setminus F$, then $E_{i,j} = 0$. I.e., sets so that all the ones in E are isolated to at most $|D|$ rows and $|F|$ columns. We let the *error isolation value* $\mu(E)$ be the smallest integer B such that there exists error isolating sets D and F with $|D|, |F| \leq B$. We show that the error isolation value is less than or equal the independent error value.

Lemma 5.2. $\mu(E) \leq \nu(E)$.

Proof. If E contains only zeros, then $\mu(E) = 0 = \nu(E)$. Otherwise, we prove the lemma by constructing a set of independent errors G from E, so that $\mu(E) \leq |G|$: pick (r_1, c_1) to be in G such that $E_{r_1,c_1} = 1$. Then greedily pick values (r_i, c_i) such that $E_{r_i,c_i} = 1$ and such that r_i is different from all $r_{j<i}$ and c_i is different from all $c_{j<i}$, and put these in G. Do this till it is not longer possible to pick a new pair (r_i, c_i). Then by definition of ν we have $|G| \leq \nu(E)$ (since G is a set of independent errors). Furthermore, note that all errors are now isolated to either rows indexed by r_i or columns indexed by c_i for some pair $(r_i, c_i) \in G$. Thus by definition of μ we have $\mu(E) \leq |G|$. □

Finally we show that an pair of almost optimal error isolating sets can be found efficient by simulator.

Lemma 5.3. *When $\mu(E) \leq \sigma$, the simulator can efficiently compute D and F with $|D| \leq \sigma$ and $|F| \leq n\sigma + \sigma$ such that if $i \in [n] \setminus D$ and $j \in [2\ell] \setminus F$, then $E_{i,j} = 0$. I.e., all the errors in the codewords are isolated to the σ rows of D and $n\sigma + \sigma$ columns of F.*[2]

Proof. To construct F and D proceed as follows: First add the index of any column in E with Hamming weight more than σ to F and add the index of any

[2]Given enough time the simulator could get $|F|$ down to σ, but we conjecture that it is NP hard to get get $|F|$ down to $O(\sigma)$ due to the similarity to the bipartite clique problem.

row with Hamming weight more than σ to D. There are at most σ such columns and σ such rows, or we could not have $\mu(E) \leq \sigma$. Now all remaining rows have at most Hamming weight σ. There are at most n remaining rows. This gives a total of $n\sigma$ non-zero entries in the remaining rows. Add the index of any columns with one of these non-zero entries to F. This gives $|F| \leq n\sigma + \sigma$. □

5.5 Extending $\mathcal{F}_{\text{WCOM}}$ For Secure Two-Party Computation

In this section we give few simple modifications to $\mathcal{F}_{\text{WCOM}}$ and π_{WCOM} that we will need for our concrete use of $\mathcal{F}_{\text{WCOM}}$ in Chapter 6.

First, as we want to treat the commitments as an ideal functionality, we need to push into the ideal functionality two extra commands (in a way similar to the commit-and-prove functionality in [CLOS02]). The first one allows to perform *Oblivious-Opening* i.e, B can choose between two sets and learn the XOR of the committed messages in one of them, without learning anything about the other set. Additionally, we need a command to allow A to open the XOR of committed messages in one out of two sets to B without revealing which one. In other words, A proves that one of the two sets of committed messages XORs to the opened message, without revealing which one. We call this an *Or-Opening*. We describe the $\mathcal{F}_{\text{WCOM}}$ functionality with these commands added in Figure 5.3, and we will need a protocol that implements this extended functionality. Note, that for technical reasons there can at most be ω Or-Openings and all Or-Openings must be done before the first Oblivious-Opening. The parameter ω can be any polynomial of the security parameter but it must be known when setting up the functionality.

Second, we need to strengthen the protocol to have only $K = \sigma$ wildcard commitments instead of $K' = O(\sigma^2)$ as for the protocol described above. I.e. we will need a protocol that implements $\mathcal{F}_{\text{WCOM}}^{K',\ell}$ instead of $\mathcal{F}_{\text{WCOM}}^{K,\ell}$.

5.5.1 Oblivious-Opening

The Oblivious-Opening can be implemented very easily in the $\mathcal{F}_{\text{OT}}$-hybrid model from any protocol implementing $\mathcal{F}_{\text{WCOM}}$ with a non-interactive opening. That is, any protocol that implements the opening part of $\mathcal{F}_{\text{WCOM}}$ by sending a single message o from A to B: A inputs as messages to $\mathcal{F}_{\text{OT}}$ the openings o_{J_0} and o_{J_1} corresponding to the sets of commitments J_0 and J_1. B inputs b to $\mathcal{F}_{\text{OT}}$ and receives o_{J_b}, which he can then open in the regular way.

This solution has an inherent selective failure attack: a corrupt A could compute one opening honestly and the other in a way that is sure to make B reject the opening. This way A will either learn b or she will be caught. However, in our concrete use of $\mathcal{F}_{\text{WCOM}}$ in Chapter 6 it turns out that we can live with this attack. Therefore, instead of trying to handle the attack we make it explicit in the $\mathcal{F}_{\text{WCOM}}$ functionality that such attacks are allowed for a corrupted A.

Init

As for $\mathcal{F}_{\text{WCOM}}$.

Commit

As for $\mathcal{F}_{\text{WCOM}}$.

Open

As for $\mathcal{F}_{\text{WCOM}}$.

Oblivious-Opening

On input $(\texttt{OT-choose}, otid, b)$ with $b \in \{0,1\}$ from B output $(\texttt{OT-choose}, otid)$ to A. On input $(\texttt{OT-open}, otid, J_0, J_1)$ from A with $J_0, J_1 \subset ID$ where for all $j \in J_0, J_1$ a pair (j, m_j) is stored and $(\texttt{OT-choose}, otid, *)$ was input before by B do the following:

- If A is honest, output $(\texttt{OT-open}, otid, J_b, \oplus_{j \in J_b} m_j)$ to B.
- If A is corrupted, wait for A to input $(\texttt{guess}, g)$ with $g \in \{0, 1, \perp\}$. If $g \in \{0,1\}$ and $g \neq b$ output `Alice cheats` to B and terminate. Otherwise, proceed to wait for A to input $(\texttt{corrupt-open}, J_0, J_1, m_{J_0}, m_{J_1})$. Add J_b to $\mathcal{J}$ and associate the equation $\oplus_{j \in J_b} \mathtt{X}_j = m_{J_b}$ to J_b. Check that the equation system still has a solution as described above. If so, output $(\texttt{OT-open}, J_b, m_{J_b})$ to B. Otherwise output `Alice cheats` to B.

OR Open

For up to ω Or-Openings, that must all occur before the first Oblivious-Opening, do the following: On input $(\texttt{OR-open}, J_0, J_1, a)$ from A, with $J_0, J_1 \subset ID, a \in \{0,1\}$ where for all $j \in J_0, J_1$ a pair (j, m_j) is stored do the following:

- If A is honest, output $(\texttt{OR-open}, J_0, J_1, \oplus_{j \in J_a} m_j)$ to B.
- If A is corrupted, and if $J_a \cap W \neq \emptyset$, wait for corrupt A to input $(\texttt{corrupt-open}, J_a, m_{J_a})$, add J_a to $\mathcal{J}$ and associate $\oplus_{j \in J_a} \mathtt{X}_j = m_{J_a}$ to J_a. Check if the equation system still has a solution as described above. If so, output $(\texttt{OR-open}, J_0, J_1, m_{J_a})$ to B. Otherwise output `Alice cheats` to B.

Figure 5.3: The ideal functionality $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega}$ for ℓ weak commitments with ω Or-Openings.

Note also that the actual Oblivious-Opening described in Figure 5.3 is of a form where B first inputs his bit b and then only later does A instruct $\mathcal{F}_{\text{WCOM}}$ to open the XOR of J_b to B. This is needed for our concrete use of $\mathcal{F}_{\text{WCOM}}$ in Chapter 6. Again this can be implement very simply: A One Time Pads (OTP) the openings before she inputs them to $\mathcal{F}_{\text{OT}}$. Later when she wants to open to B she sends both OTP keys to B, which allows B to recover o_{J_b}.

We give a formal description of the implementation of $\mathcal{F}_{\text{WCOM}}$ extended with the Oblivious-Opening in Figure 5.4, and we prove the following.

Theorem 5.2. *Let σ be the security parameter, $K = \sigma n + \sigma$ and let* ssecc *be a code with $n = \Theta(\sigma)$, $t = \Theta(n)$, $d = \Theta(n)$ and $\sigma < d/2$ as, e.g., given by [CC06]. Then the protocol in Figure 5.4 UC, active, static securely implements $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ with the Oblivious-Opening command of Figure 5.3 in the $(\mathcal{F}_{\text{OT}}, \binom{n}{t}\text{-}\mathcal{F}_{\text{ROT}}(2\ell))$-hybrid model.*

Proof. We focus on the simulator when simulating the Oblivious-Opening command, since for all other commands we can use the simulator from the proof of Theorem 5.1. Furthermore, as above the case of corrupted B is trivial so we

Setup

As in Figure 5.2.

Commit

As in Figure 5.2.

Open

As in Figure 5.2.

Oblivious-Opening

To do an Oblivious-Opening with the sets of commitments J_0 and J_1 the parties does the following.

1. A computes openings o_{J_0} and o_{J_1} of the XOR of commitments in the sets J_0 and J_1 respectively as she would for regular openings described in Figure 5.2. She then samples $M_0 \in_R \{0,1\}^{|o_{J_0}|}$ and $M_1 \in_R \{0,1\}^{|o_{J_1}|}$ (where $|x|$ denotes the bit-length of x), and inputs to $(M_0 \oplus o_{J_0}, M_1 \oplus o_{J_1}) = (O_0, O_1)$ to $\mathcal{F}_{\text{OT}}$.
2. B inputs b to $\mathcal{F}_{\text{OT}}$ and receives $O_b = M_b \oplus o_{J_b}$.
3. Later A can open the XOR of commitments J_b by sending (M_0, M_1) to B. B computes the opening $o_{J_b} = O_b \oplus M_b$ and opens as a regular opening described in Figure 5.2.

Figure 5.4: The commitment scheme extended with Oblivious Openings

will focus on corrupted A.

On output $(\texttt{OT-choose}, otid)$ from $\mathcal{F}_{\text{WCOM}}$ the simulator observes the messages (O'_0, O'_1) as A inputs them to $\mathcal{F}_{\text{OT}}$. Later when A sends OTP keys M_0 and M_1 the simulator computes both openings $o'_0 = O'_0 \oplus M_0$ and $o'_1 = O'_1 \oplus M_1$. For each $i \in \{0,1\}$ the simulator parses o'_i as $o'_i = (x_{J_i}, r_{J_i}, J_i)$ and inputs $(\texttt{OT-open}, otid, J_0, J_1)$ to $\mathcal{F}_{\text{WCOM}}$. So far the simulation is perfect.

Note that from this point on the behaviour of B in the real world protocol is the same as when he receives o'_b as the opening in a regular opening. Since there is no other communication from B to A, A will only learn whether or not B accepts or rejects the opening of o'_b. Apart from handling the input $(\texttt{guess}, g)$, essentially the same goes for the ideal functionality $\mathcal{F}_{\text{WCOM}}$. Thus, if the simulator ignores o'_{1-b} and runs the simulator from the proof of Theorem 5.1 when opening o'_b, the simulation will go through. The only problem is that the simulator does not know the value of b. To solve this the simulator will run the simulator from the proof of Theorem 5.1 on both o'_0 and o'_1, and if necessary use $\mathcal{F}_{\text{WCOM}}$ to try and guess b. There are three cases:

1. The simulator of Theorem 5.1 rejects both openings. In this case the simulator of the Oblivious-Opening can safely reject and terminate, since o'_b must be a rejecting opening.

   The simulation is clearly perfect.

2. The simulator of Theorem 5.1 accepts both openings. In this case o'_b must be an accepting opening. The simulator of the Oblivious-Opening can safely input $(\texttt{guess}, \perp)$ followed by $(\texttt{corrupt-open}, J_0, J_1, m_{J_0}, m_{J_1})$ to $\mathcal{F}_{\text{WCOM}}$, were m_{J_0} and m_{J_1} are computed as by the simulator of Theorem 5.1.

The simulation is indistinguishable from the real world by the same argument as in the proof of Theorem 5.1.

3. For some $g \in \{0,1\}$ the simulator of Theorem 5.1 rejects on opening o'_{1-g} but not o'_g. In this case the simulator of the Oblivious-Opening inputs $(\texttt{guess}, g)$ to $\mathcal{F}_{\text{WCOM}}$. If $\mathcal{F}_{\text{WCOM}}$ terminates the simulator does the same. Otherwise, the simulator learns that $b = g$, and proceeds to input $(\texttt{corrupt-open}, J_0, J_1, m_{J_0}, m_{J_1})$ where m_{J_b} is computed as by the simulator of Theorem 5.1 and $m_{J_{1-b}} = 0^\psi$.

   If $\mathcal{F}_{\text{WCOM}}$ terminates on input $(\texttt{guess}, g)$ the simulation is perfect since o'_b is a rejecting opening. Otherwise, the simulation is indistinguishable from the real world by the same argument as in the proof of Theorem 5.1.

So since in all cases the simulation is indistinguishable from the from the real world protocol, this concludes our proof.

□

5.5.2 Less Wildcards

We now show how our scheme can be strengthened to only have $K = \sigma$ wildcard commitments, instead of $K' = O(\sigma^2)$ as in the protocol described in Figure 5.2. We do this in a black box way using a $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ to implement a $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ functionality: A uses $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ to commit to 2ℓ random values x_i. Then we randomly pair all the commitments into ℓ pairs, open the XOR of each pair, and use the left element in each pair as the commitment of $\mathcal{F}_{\text{WCOM}}^{K,\ell}$, sending just a correction value y_i as the actual commitment (similar to the protocol in Figure 5.2). The idea being that if any wildcard commitment i is paired with a non-wildcard commitment j it will become fixed by opening the XOR of the pair. Therefore, the resulting commitments are only wildcard if they were in a pair where both commitments were wildcard commitments. If we take $\ell > K'^2$, then the probability that there are more than σ such pairs of wildcard commitments is negligible. We describe the protocol formally in Figure 5.5 and we prove the following theorem.

Theorem 5.3. *Let σ be the security parameter, $K = \sigma$ and $\ell > K'^2$. Then the protocol in Figure 5.5 UC, active, static securely implements $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ in the $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ hybrid model.*

Proof. In the setup phase the simulator behaves as an honest B while fully controlling the $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ functionality. Doing so the simulator can read of the set of wildcard commitments input by corrupt A W' when initializing $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$. For all inputs $(\texttt{commit}, i, x_i)$ made by A during the setup the simulator records (x_i, i). For all inputs $(\texttt{open}, \{i, \pi(i)\})$ followed by $(\texttt{corrupt-open}, \{i, \pi(i)\}, z_i)$ made by A the simulator records (z_i, i).

After the setup phase the simulator computes the set $W = \{i \in ID \cap W' | \pi(i) \in W'\}$, i.e., W is the set of wildcard commitments in ID that were paired with another wildcard commitments. To initialize the $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ functionality the simulator then inputs ID as the set of commitment identifiers and W

Setup

1. A and B initialize a $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ functionality which outputs ID' to both parties.
2. For all $i \in [2\ell]$ A samples $x_i \in_{\text{R}} \{0,1\}^{\psi}$ and inputs $(\texttt{commit}, i, x_i)$ to $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$.
3. After receiving $(\texttt{commit}, i)$ for all $i \in [2\ell]$ from $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ B picks a uniformly random pairing π (a permutation $\pi : ID' \to ID'$ where $\forall i, \pi(\pi(i)) = i$) and sends it to A.
4. A sets $ID = \{i \in ID' | i < \pi(i)\}$. For all $j \in ID = \{i \in [2\ell] | i < \pi(i)\}$ A inputs $(\texttt{open}, \{i, \pi(i)\})$ to $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ and outputs ID as the set of commitment id's.
5. On receiving $(\texttt{open}, \{i, \pi(i)\}, z_i = x_i \oplus x_{\pi(i)})$ for all $i \in ID$ B outputs ID.
6. For the remainder of the protocol A is not allowed to use id's in $ID' \setminus ID$. If she does so B terminates the protocol.

Commit

To commit to $m_j \in \{0,1\}^{\psi}$ for $j \in ID$ A sends $(y_j = m_j \oplus x_j, j)$ to B.

Open

To open the XOR of commitments $J \subset ID$ the parties does the following.

1. A inputs $(\texttt{open}, J)$ to the $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ functionality.
2. On output $(\texttt{open}, J, x_J)$ from $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ B computes $m_J = (\bigoplus_{j \in J} y_j) \oplus x_J$ and outputs $(\texttt{open}, J, m_J)$.

Oblivious-Opening

A and B use the **Oblivious-Opening** command of the $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ functionality. On output $(\texttt{OT-open}, J_b, x_{J_b})$ from $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ B computes $m_{J_b} = (\bigoplus_{j \in J_b} y_j) \oplus x_J$ and outputs $(\texttt{OT-open}, J_b, m_{J_b})$.

Figure 5.5: The commitment scheme with less wildcards.

as the set of wildcard commitments. This makes $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ output ID to B and simulation is clearly perfect.

From this point on the simulator is going to simulate the $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ functionality by essentially just forwarding messages between A and $\mathcal{F}_{\text{WCOM}}^{K,\ell}$. Note though that if A gives an input to $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ with an id $j \in ID' \setminus ID$ then this input can not simply be forwarded to $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ because that functionality only knows about id's in ID. This is not a problem though because for any such input A could make, $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ outputs j to B and honest B terminates. So, in case of such an input the simulator just terminates the protocol. In the following we will assume that A only makes inputs to $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ with id's in ID.

To simulate the j'th commitment the simulator records the value (y_j, j) sent by A. If $j \in W' \setminus W$ the simulator lets $m_j = (x_{\pi(j)} \oplus z_j) \oplus y_j$. Otherwise, the simulator lets $m_j = x_j \oplus y_j$. Then the simulator inputs $(\texttt{commit}, j, m_j)$ to the $\mathcal{F}_{\text{WCOM}}^{K,\ell}$, and the functionality outputs $(\texttt{commit}, j)$ to B.

Note that for $j \in ID \setminus W'$ the $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ functionality used in the ideal world stores the equation $\mathtt{X}_j = x_j \oplus y_j$ while the $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ functionality used in the real world would have stored $\mathtt{X}_j = x_j$. Furthermore, for $j \in W' \setminus W$ $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ stores $\mathtt{X}_j = x_{\pi(j)} \oplus z_j \oplus y_j$ while $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ stores $\mathtt{X}_j \oplus \mathtt{X}_{\pi(j)} = z_j$ and $\mathtt{X}_{\pi(j)} = x_{\pi(j)}$. Thus,

ignoring the correction values, the restrictions on how to open commitment $j \in ID$ is equivalent in both worlds. I.e. the XOR of commitments in any set $J \subset ID$ can, in the real world, be opened to x_J using $\mathcal{F}_{\text{WCOM}}^{K',2\ell}$ if and only if the XOR of commitments in J can be opened to $m_J = x_J \oplus (\bigoplus_{j \in J} y_j)$ in the ideal world using $\mathcal{F}_{\text{WCOM}}^{K,\ell}$.

On input $(\texttt{open}, J)$ from A the simulator forwards the input to $\mathcal{F}_{\text{WCOM}}^{K,\ell}$. On input $(\texttt{corrupt-open}, J, x_J)$ the simulator inputs $(\texttt{corrupt-open}, J, m_J = x_J \oplus (\bigoplus_{j \in J} y_j))$ to $\mathcal{F}_{\text{WCOM}}^{K,\ell}$. By the discussion above the simulation is perfect.

To simulate the Oblivious-Openings the simulator forwards messages between A and the $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ in essentially the same way as for the regular opening. I.e. all messages are forwarded verbatim except if A inputs the message $(\texttt{corrupt-open}, J_0, J_1, x_{J_0}, x_{J_1})$. In that case the simulator inputs to $\mathcal{F}_{\text{WCOM}}$ $(\texttt{corrupt-open}, J_0, J_1, x_{J_0} \oplus (\bigoplus_{j \in J_0} y_j), x_{J_1} \oplus (\bigoplus_{j \in J_1} y_j))$. Again the simulation is perfect.

The only thing left to argue is that with overwhelming probability $|W| \leq K = \sigma$. Consider any of ℓ pairs of commitments opened in the setup phase. The probability that both commitments in this pair are wildcard commitments less than $(K'/2\ell)^2$. As there are ℓ pairs, by union bound we have

$$\begin{aligned}
\Pr(|W| \geq K) &= \binom{\ell}{K} \left(\frac{K'}{2\ell}\right)^{2K} \\
&\leq \ell^K \left(\frac{K'}{2\ell}\right)^{2K} \\
&\leq K'^{2K} K'^{-2K} 2^{-2K} = 2^{-2K} \ ,
\end{aligned}$$

where the last inequality is by $\ell > K'^2$. I.e for K in $\omega(\sigma)$ we have that $\Pr(|W| \geq K)$ is negligible, and this includes the case where $K = \sigma$. □

5.5.3 Or-Opening

The Or-Opening can be implemented in a black box way given the $\mathcal{F}_{\text{WCOM}}$ functionality (without the Or-Opening command): A sends $m = \oplus_{i \in J_a} m_i$ to B, and then proves that it is a correct value, i.e. that $m = \bigoplus_{i \in J_0} m_i \vee m = \bigoplus_{i \in J_1} m_i$. To do this, she makes two new commitments, to $m_{J_0} = \bigoplus_{j \in J_0} m_j$ respectively $m_{J_1} = \bigoplus_{j \in J_1} m_j$. She will do this by sampling a bit $p \in \{0, 1\}$ and then use $\mathcal{F}_{\text{WCOM}}$ to commit to values $m'_0 = m_{J_{0 \oplus p}}$ and $m'_1 = m_{J_{1 \oplus p}}$ without revealing p to B. Then B will challenge A with a bit c. If $c = 0$, A reveals p and uses $\mathcal{F}_{\text{WCOM}}$ to open $(\oplus_{j \in J_0} m_j) \oplus m_{J_0}$ to 0 and $(\oplus_{j \in J_1} m_j) \oplus m_{J_1}$ to 0, proving that the messages m_{J_0} and m_{J_1} where computed correctly. If $c = 1$, A opens the commitment to $m'_{a \oplus p} = m$. If we disregard wildcard commitments, this is a zero-knowledge proof with soundness $\frac{1}{2}$. By repeating $O(\sigma)$ times we get negligible soundness error.

However, if A commits to m'_0 and m'_1 using wildcard commitments she can open the commitments as she pleases, and soundness drops to 0. To deal with this, we let B randomly chose the indices used by A to commit to m'_0 and m'_1. Now with high probability there will be $O(\sigma)$ repetitions where neither m'_0 nor m'_1 are committed to using wildcard commitments.

Setup

A and B initialize a $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ functionality. Additionally B sends to A ω uniformly random but disjoint sets $D_1, \ldots, D_\omega \subset ID$ so that each D_i has size 6σ. These id's are set aside only for use in the Or-Opening.

Commit

A uses the **Commit** command of the $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ functionality.

Open

A uses the **Open** command of the $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ functionality.

Oblivious-Opening

A and B use the **Oblivious-Opening** command of the $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ functionality.

Or-Opening

To perform the d'th Or-Opening the parties do the following (recall that no Or-Openings are done after the first Oblivious-Opening).

1. A computes $m_{J_0} = \oplus_{j \in J_0} m_j$ and $m_{J_1} = \oplus_{j \in J_1} m_j$ and sends $(J_0, J_1, m = m_{J_a})$ to B.
2. B parses D_d as $D_d = \{j_0^1, j_1^1, \ldots j_0^{3\sigma}, j_1^{3\sigma}\} \subset ID$.
3. For each $i \in [3\sigma]$ A samples $p^i \in_{\text{R}} \{0,1\}$ and inputs $(\texttt{commit}, j_0^i, m_{J_{0 \oplus p^i}})$ and $(\texttt{commit}, j_1^i, m_{J_{1 \oplus p^i}})$ to $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$.
4. After receiving $(\texttt{commit}, j_0^i)$ and $(\texttt{commit}, j_1^i)$ from $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ for all $i \in [3\sigma]$, B sends $(c^i)_{i \in [3\sigma]} \in_{\text{R}} \{0,1\}^{3\sigma}$ to A.
5. For each $i \in [3\sigma]$ A does the following:
   - If $c^i = 0$: A inputs $(\texttt{open}, J_{0 \oplus p^i} \cup \{j_0^i\})$ and $(\texttt{open}, J_{1 \oplus p^i} \cup \{j_1^i\})$ to $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ and sends p^i to B.
   - If $c^i = 1$: A inputs $(\texttt{open}, \{j_{a \oplus p^i}^i\})$ to $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$.
6. For each $i \in [3\sigma]$ B does the following:
   - If $c^i = 0$: B checks that he receives both $(\texttt{open}, J_{0 \oplus p^i} \cup \{j_0^i\}, 0)$ and $(\texttt{open}, J_{1 \oplus p^i} \cup \{j_1^i\}, 0)$ from $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$.
   - If $c^i = 1$: B checks that he receives $(\texttt{open}, \{j^i\}, m)$ from $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ for some $j^i \in \{j_0^i, j_1^i\}$.

   If any of the checks fail B rejects the opening and terminates the protocol. Otherwise, B outputs $(\texttt{open}, J_0, J_1, m)$.

Figure 5.6: The commitment scheme extended with Oblivious and Or Openings

We formally describe the protocol used to implement $\mathcal{F}_{\text{WCOM}}$ with the or opening in Figure 5.6 and prove the following theorem.

Theorem 5.4. *Let σ be the security parameter, $\ell > 6K\sigma$ and $\ell' = \ell + 3\sigma\omega$, where $\omega = \text{poly}(\sigma)$ is the number of Or-Openings. Then the protocol in Figure 5.6 UC, active, static securely implements $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega}$ with the Or-Opening command in the $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ hybrid model.*

Proof. For all commands apart from the Or-Opening the simulator simulates all interaction between A and $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ by interacting correspondingly with the $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega}$ functionality and recording all messages. Clearly this simulation is perfect, so for the remainder of proof we will concentrate on the Or-Opening command.

For the Or-Opening the simulator acts as the honest B. Note that since the simulator has recorded all messages between A and $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ it can simulate the

$\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ functionality perfectly. This means that when the simulator rejects the opening the simulation has been perfect.[3]

Assuming that the opening was not rejected, the simulator then needs to interact with $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega}$ so that the functionality outputs $(\texttt{OR-open}, J_0, J_1, m)$ to B with overwhelming probability. I.e. the simulator must find an a' so that either $J_{a'} \cap W = \emptyset$ and $\bigoplus_{j \in J_{a'}} m_j = m$ or $J_{a'} \cap W \neq \emptyset$ and the equations stored in the $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega}$ functionality allows to corruptly open the XOR of commitments $J_{a'}$ to m.

To show that the simulator can do this we first show that, given the opening was not rejected, there exists an $i' \in [3\sigma]$ and a $j_{a'}^{i'} \in \{j_0^1, j_1^1, \ldots, j_0^{3\sigma}, j_1^{3\sigma}\} = D_d$ with the properties that

1. $m_{a^{i'}}^{i'} = m$, where $m_{a'}^{i'}$ is the message committed to as commitment $j_{a'}^{i'}$.

2. The XOR of commitments in $J_{a' \oplus p^{i'}}$ can be opened to $m_{a'}^{i'}$ (honestly or corruptly).

Notice that such a $j_{a'}^{i'}$ implies that $J_{a' \oplus p^{i'}}$ can be opened to m.

We would like to prove this by showing that for each $i \in [3\sigma]$ if there is not a $j_{a'}^{i}$ with both properties then the opening is rejected with probability $\frac{1}{2}$. Thus, assuming the opening is not rejected the probability that there is no $i \in [3\sigma]$ with a $j_{a'}^{i}$ with both properties is at most $2^{-3\sigma}$. However, the wildcard commitments get in the way of such a straight forward proof.

Instead we assume that the set $H \subset [3\sigma]$ where for all $i \in H$ neither j_0^i nor j_1^i is a wildcard commitment, i.e.

$$H = \{i \in [3\sigma] | j_0^i, j_1^i \notin W\} \ ,$$

has size least σ. Below we show that since honest B chooses the set $D_d \subset ID$ uniformly at random, this is the case with overwhelming probability.

We then have that for each $i \in H$ the opening will reject with probability at least $\frac{1}{2}$ if there is not a $j_{a'}^i$ with property 1. Namely, if $c^i = 1$ A must open a commitment $j_{a'}^i$ to m and by $i \in H$ we have that $j_{a'}^i \notin W$ can only be opened to $m_{a'}^i$.

On the other hand for each $i \in H$ the opening will also reject with probability at least $\frac{1}{2}$ if there is not a $j_{a'}^i$ with property 2. Namely, if $c^i = 0$ A must be able to open the XOR of commitments in $J_{a' \oplus p^i} \cup \{j_{a'}^i\}$ to 0, but since $j_{a'}^i \notin W$ this is equivalent to being able to open the XOR of commitments in $J_{a' \oplus p^i}$ to $m_{a'}^i$.

Thus with probability at least $1 - 2^{-\sigma}$ there exists an $i \in H$ and a $j_{a'}^i$ with both the above properties. Note that since the simulator knows all messages sent to $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ by A it can easily find such a $j_{a'}^i$.

[3]Note that to simulate the $\mathcal{F}_{\text{WCOM}}^{K,\ell'}$ functionality the simulator needs to know all the equations stored inside the functionality. This is why we cannot do Or-Openings after the first Oblivious-Opening: After an Oblivious-Opening the equations stored in $\mathcal{F}_{\text{WCOM}}$ depend on B's secret bit b which would be unknown to the simulator.

The simulator then inputs $(\texttt{OR-open}, J_0, J_1, a')$ to the $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ functionality, and if $J_{a'} \cap W \neq \emptyset$ proceeds to input $(\texttt{corrupt-open}, J_{a'}, m)$. The $\mathcal{F}_{\text{WCOM}}^{K,\ell}$ functionality then outputs (J_0, J_1, m) to B and the simulation is indistinguishable to the real world protocol.

To conclude the proof we must show that with overwhelming probability $|H| \geq \sigma$. To see this we use that if $|D_d \cap W| \leq s$ then $|H| \geq 3\sigma - 2s$. Thus we will show that $|D_d \cap W| \leq \sigma$ with overwhelming probability.

Consider some $j \in D_d$ and assume

$$(\bigcup_{i \in [\omega]} D_i \setminus \{j\}) \cap W = \emptyset \ ,$$

then the probability that $j \in W$ is $K/(\ell' - 6\sigma\omega + 1) = K/(\ell + 1)$. Therefore, for each $j \in \{j_0^1, j_1^1, \ldots, j_0^{3\sigma}, j_1^{3\sigma}\}$ we have that

$$\Pr(j \in W) \leq \frac{K}{\ell + 1} \ .$$

Now, let $Y_1, \ldots, Y_{6\sigma}$ be random independent variables where each $Y_i = 1$ with probability $K/(\ell + 1)$ and $Y_i = 0$ otherwise, and let $S = \sum_{i \in [6\sigma]} Y_i$. We then have that

$$1 - \Pr(|H| \geq \sigma) \leq \Pr(|D_d \cap W| > \sigma|) \leq \Pr(S > \sigma|) \ .$$

The random variable S has expected value $\mathrm{E}[S] = 6\sigma \frac{K}{\ell+1} \leq 1$ by assumption on ℓ. Thus by Hoeffdings inequality we have that

$$\Pr(S > \sigma) \leq e^{-2\frac{2\sigma^2}{6\sigma}} = e^{-\frac{2}{3}\sigma} \ ,$$

which means that

$$\Pr(|H| \geq \sigma) \geq 1 - e^{-\frac{2}{3}\sigma} \ .$$

I.e., we have shown that H has size at least σ with overwhelming probability. □

Finally combining the results of this chapter we get the following.

Corollary 5.1. *Let σ be the security parameter, $\ell > (\sigma n + \sigma)^2 > 6\sigma^2$, $\omega = \text{poly}(\sigma)$ and $K = \sigma$, and assume a code* ssecc *with $n = \Theta(\sigma)$, $t = \Theta(n)$, $d = \Theta(n)$ and $\sigma < d/2$ as, e.g., given by [CC06]. Then $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega}$ with Oblivious- and Or-Openings can be UC, active, static securely implemented in the $(\binom{n}{t}\text{-}\mathcal{F}_{\text{ROT}}(4\ell + 12\sigma\omega), \mathcal{F}_{\text{OT}})$-hybrid model.*

Proof. By Theorem 5.2 we can implement $\mathcal{F}_{\text{WCOM}}^{K',2(\ell+6\sigma\omega)}$ with the Oblivious-Opening in the $(\binom{n}{t}\text{-}\mathcal{F}_{\text{ROT}}(4\ell + 12\sigma\omega), \mathcal{F}_{\text{OT}})$-hybrid model, with $K' = \sigma n + \sigma$. By Theorem 5.3 and $\ell > K'^2$ we can use $\mathcal{F}_{\text{WCOM}}^{K',2(\ell+6\sigma\omega)}$ to implement $\mathcal{F}_{\text{WCOM}}^{K,\ell+6\sigma\omega)}$ with the Oblivious-Opening. Finally by Theorem 5.4 and $\ell > 6\sigma^2$ we can use $\mathcal{F}_{\text{WCOM}}^{K,\ell+6\sigma\omega)}$ to implement $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega)}$. □

5.6 Complexity Analysis

We sketch a complexity analysis counting the calls to the symmetric primitives used in the protocol for $\mathcal{F}_{\text{WCOM}}^{K,\ell,\omega}$. This includes the encoding algorithm of the code ssecc used (notice that the protocol never decodes), and H. We count the total number of calls made by A and B.

5.6.1 Base Protocol

The complexity of the base protocol, as described in Figure 5.2, is dominated by the **Setup**-phase: here we do 2ℓ random commitments and open ℓ to check for correctness. In addition we need to do one $\binom{n}{t}$-$\mathcal{F}_{\text{ROT}}(2\ell)$. Committing requires one encoding of A and opening requires one encoding of B. I.e. we get 3ℓ calls to enc during **Setup**. The $\binom{n}{t}$-$\mathcal{F}_{\text{ROT}}(2\ell)$ functionality can be implemented using $\binom{n}{t}$-$\mathcal{F}_{\text{ROT}}(\psi)$ as described in Chapter 2. This requires $\frac{2\ell(n+t)}{\psi}$ calls to H. The $\binom{n}{t}$-$\mathcal{F}_{\text{ROT}}(\psi)$ can be implemented using $t\log(n)$ ROTs. Using the techniques of Chapter 3 this costs 14 calls to H per ROT and $O(\psi)$ seed OTs.

For the **Commit** part of the protocol no calls are made to H or enc, as we are simply sending the correction values. For the **Open** part B needs to do a single encoding to compare with his watch bits.

In total we get

$$3\ell + \gamma$$

calls to enc, and

$$\frac{2\ell(n+t)}{\psi} + 14t\log(n)$$

calls to H, where γ is the number of openings, n is the length of codewords and t is the amount of watch bit positions. Note, that n and t are highly dependent on the code we use to implement the encoding ssecc.

5.6.2 Extended Protocol

To implement $\mathcal{F}_{\text{WCOM}}$ with σ instead of $O(\sigma^2)$ wildcards the cost of **Setup** doubles, additionally we need to do ℓ openings to make sure that most wildcards get fixed. The cost of the **Commit** and **Open** parts of the protocol are unchanged. I.e. $\mathcal{F}_{\text{WCOM}}$ with the decreased wildcards extension requires

$$7\ell + \gamma$$

calls to enc, and

$$\frac{4\ell(n+t)}{\psi} + 14t\log(n)$$

calls to H.

Oblivious-Opening essentially has the same cost as a regular opening, except that it requires one OT, which can be implemented with 14 calls to H.

Extending with **Or-Opening** requires us to do **Setup** for an additional $6\sigma\omega$ commitments, where ω is the number of Or-Openings we need to do. Doing the

Or-Opening it self requires 6σ openings each using one encoding. I.e. with all the extensions the cost of $\mathcal{F}_{\text{WCOM}}$ is

$$7\ell + 8(6\sigma\omega) + \gamma = 7\ell + 48\sigma\omega + \gamma$$

calls to enc, and

$$\begin{aligned}&\frac{(4\ell + 4(6\sigma\omega))(n+t)}{\psi} + 14t\log(n) + 14\lambda \\ &= \frac{(4\ell + 24\sigma\omega)(n+t)}{\psi} + 14t\log(n) + 14\lambda\end{aligned}$$

calls to H, where λ is the number of Oblivious-Openings. In other words we do 7 calls to enc and $4(n+t)/\psi$ calls to H for each commitment, and 1 call to enc for each opening. For each Or-Opening we need an additional 42σ calls to enc and $24\sigma(n+t)/\psi$ calls to H in **Setup** and 6σ calls to enc in the actual opening. For each Oblivious-Opening we need an additional 14 calls to H. To setup the protocol we need $14t\log(n)$ calls H and $O(\psi)$ seed OT's.

Thus the efficiency of the scheme is highly dependent on the size of n and t, which again is dependent on the encoding used in the scheme. Below we discuss some choices of this code.

5.6.3 Encoding

The main difficulty in picking an encoding is that we need the size of codewords n to be at most a constant times σ for as small a constant as possible. Using the highly efficient schemes of [CC06], we estimate that we get an encoding where $n = 40\psi$ and $t = \psi$. Assuming $\psi = c\sigma$ for some small c this achieves our purpose. Suppose, for example, we use computational security parameter $\psi = 120$ and statistical security parameter $\sigma = 80$. Then this means that we get codewords of size 4800 bits and will need approximately 164 calls to H and for each commitment, and ~ 1500 initial calls to H to set up the protocol.

Alternatively, we can essentially use any linear secret sharing scheme with $t = \sigma$ privacy: We let the codeword of a message $m \in \{0,1\}^{\psi}$ be the concatenation of 3σ shares. It is easy to verify that this gives us the linearity, and privacy we need, and that encoding in this way gives us minimum distance 2σ. For example, we could use Reed Solomon codes (essentially Shamir sharing) over the field $\mathbb{F}_{2^9}$, viewing each bit of the message $m \in \{0,1\}^{\psi}$ as an element in the field. This would give us codewords of size $9 \cdot 3\sigma = 2160$ bits and would require 74 calls to H per commitment.

The problem with this approach is that the field needs to grow logarithmically with the security parameter, witch would require us to adjust our security proof. However, for most practical parameters this solution should be fine.

Chapter 6

MiniLEGO

In this chapter we present the MiniLEGO protocol. The idea behind protocol is essentially that of the LEGO protocol of Nielsen and Orlandi [NO09]: we generate many independent garbled gates for Yao's semi-honest protocol. We then do a cut-n-choose test on the gates to test that most of them are honestly generated. Finally we connect the garbled gates in a fault tolerant circuit that securely evaluates the intended circuit even if a few of the gates are faulty. In building this circuit we need a way to *solder* the independently generated garbled gates to each other. To this end we need a homomorphic commitment scheme.

The main differences between the protocol presented here and the LEGO protocol is 1) that we replace the use of Pedersen commitments with the commitment functionality presented in Chapter 5 and 2) that this allows us to use standard garbling techniques which in turns allows us to use standard optimizations and to give a much less complex protocol.

Overview

- In Section 6.1 we essentially restate the ideal functionality of LEGO.
- In Section 6.2 we give the main building blocks for the protocol: We describe generic garbling scheme, including how to connect gates using soldering, and for self-containment we restate the commitment functionality given in Chapter 5.
- We then describe the MiniLEGO 2PC protocol in Section 6.3.
- We give the security analysis in Section 6.4 which is also quite similar to that of the LEGO protocol, however, we must be a little careful to handle the wildcard commitments that where not present in LEGO.
- Finally we sketch a complexity analysis counting the symmetric primitives used in the protocol.

6.1 The Ideal Functionality

In Figure 6.1 the ideal functionality for secure function evaluation is presented (taken almost verbatim from [NO09]). Note that the functionality is insecure in the sense that A can try to guess B's input bits, but if her guess is wrong B is told that A is cheating. This models a standard problem in Yao based protocols, that can be solved by modifying the circuit to be evaluated. For instance, to evaluate a circuit $C((a_i)_{i\in[\ell]}, (b_i)_{i\in[\ell]})$ securely one could instead evaluate $C'((a_i)_{i\in[\ell]}, (b_{i,j})_{i\in[\ell],j\in[\psi]}) = C((a_i)_{i\in[\ell]}, (\oplus_{i\in[\psi]} b_{i,j})_{j\in[\ell]})$ i.e., B encodes his real input bit in the parity of a ψ bit-long string, and the modified circuit first reconstructs the real input and then evaluates the original circuit. Now, in order to guess one of B's real input bits A needs to guess correctly the ψ random bits, so she will fail with probability $1 - 2^{-\psi}$.

As our construction allows to compute XOR gates for free, this increases only marginally the time needed to construct and evaluate the circuit. On the other hand, this increases the number of OT's needed during the input phase by a factor ψ (but OT extension can be used – so this is also not too bad). Better encodings can be used (See [LP07]) to reduce the size of the encoded input from $\ell \cdot \psi$ bits to $\max(4\ell, 8\psi)$ bits.

The ideal functionality $\mathcal{F}_{\text{SFE}}$ runs as follows:

Circuit and inputs: A inputs (a, c_A), B inputs (b, c_B), and c_A and c_B are leaked to the adversary. If $c_A \neq c_B$, then the ideal functionality outputs `disagreement!` to both parties and terminates. Otherwise, let $c = c_A$ and parse c as (ℓ, C), where $\ell \in \mathbb{N}$ and C is a circuit with 2ℓ input wires and ℓ output wires. Parse a as $a \in \{0,1\}^\ell$ and $b \in \{0,1\}^\ell$.

Corrupt A: If A is corrupted, she can specify a set $\{(i, \beta_i)\}_{i\in I}$, where $I \subseteq \{1, \ldots, \ell_B\}$ and $\beta_i \in \{0,1\}$. If $\beta_i = b_i$ for $i \in I$, then output `correct!` to A. Otherwise, output `You were nicked!` to A and output `Alice cheats!` to B.

Evaluation: If both parties are honest or A was not caught above, then the ideal functionality computes $z = f(a, b)$ and outputs z to A. The adversary decides the time of delivery.

Figure 6.1: The ideal functionality for secure circuit evaluation.

6.2 Building Blocks

Here we describe the main building blocks we use to build the protocol. First we describe a generic version of a Yao garbling scheme supporting the free-XOR construction. For a concrete scheme we refer to [BHR12]. Second we restate the commitment functionality that we implemented in Chapter 5.

6.2.1 Generic Free-XOR Yao Gate

Here we describe a generic Yao garbling scheme that supports the free-XOR construction.

- We have a (possibly randomized) algorithm Yao that on inputs a gate identifier id, a left input *zero-key* $L_0 \in \{0,1\}^\psi$, a right input zero-key $R_0 \in \{0,1\}^\psi$ and a global difference $\Delta \in \{0,1\}^\psi$ outputs a garbled gate gg and a output zero-key $O_0 \in \{0,1\}^\psi$.
- We have a (possibly randomized) algorithm Eval that on input a left key $L' \in \{0,1\}^\psi$, a right key $R' \in \{0,1\}^\psi$ and a garbled gate gg outputs an output key $O' \in \{0,1\}^\psi \cup \{\perp\}$.
- We define the *one-keys* L_1,R_1,O_1 s.t. $L_0 \oplus L_1 = R_0 \oplus R_1 = O_0 \oplus O_1 = \Delta$.

The idea is that a garbled AND gate gg has a zero- and a one-key associated with each of its wires (left input, right input and output wire), and that these keys represent the bit values on those wires. E.g. if gg is a garbled AND gate generated as $(gg, O_0) \leftarrow \mathsf{Yao}(id, L_0, R_0, \Delta)$ then $\mathsf{Eval}(gg, L_a, R_b)$ for any $a, b \in \{0,1\}$ should output $O_{a \wedge b}$.

Note that if A samples Δ and a zero-key, say L_0, at random and give the key L_a to B then there is no way for B to infer the bit a from L_a. Furthermore, even if B learns a he can not guess the key L_{1-a} with better probability than guessing Δ. For a garbling scheme to be secure we want that even if B learns gg and keys L_a and R_b for $a, b \in \{0,1\}$, and is able to evaluate $O_{a \wedge b} \leftarrow \mathsf{Eval}(gg, L_a, R_b)$, then he can not guess L_{1-a}, R_{1-b} or $O_{1-a \wedge b}$ with better probability then guessing the random string Δ, even if he knows a and/or b.

Thus B can evaluate the garbled gate gg without knowing anymore about the output than he can infer from his knowledge of a and b. Furthermore, B can not evaluate the gate on any other inputs. Thus if B sends back $O_{a \wedge b}$ to A, A can learn $a \wedge b$ (as she knows O_0 and Δ) and be confident that this is the correct result.

We formalize this intuition about correctness and security of a garbled gate in definition Def. 6.1.

Definition 6.1. We say $(\mathsf{Yao}, \mathsf{Eval})$ is a *Yao free-XOR garbling scheme* if the following holds:

Correctness: Let $(gg, O_0) \leftarrow \mathsf{Yao}(id, L_0, R_0, \Delta)$, then for all $a, b \in \{0,1\}$

$$\mathsf{Eval}(gg, L_a, R_b) = O_{a \wedge b}$$

with overwhelming probability over the choices of L_0, R_0, Δ and the random coins of Yao and Eval.

Secrecy: Consider the following indistinguishability under chosen input attack game for a stateful adversary $\mathcal{A}$.

$\mathsf{IND\text{-}CIA}^{\mathcal{A}}_{(\mathsf{Yao},\mathsf{Eval})}(\psi)$

$(a_0, b_0) \leftarrow \mathcal{A}(1^\psi)$, where $(a_0, b_0) \in \{0,1\}^{2\ell}$
$(a_1, b_1) \leftarrow \mathcal{A}(1^\psi)$, where $(a_1, b_1) \in \{0,1\}^{2\ell}$
$c \leftarrow \{0,1\}$, $\Delta \leftarrow \{0,1\}^\psi$
$(L^i, R^i) \leftarrow \{0,1\}^{2\psi}$, for $i = 1, \ldots \ell$
$d \leftarrow \mathcal{A}(\{\mathsf{Yao}(id, L^i, R^i, \Delta), L^i_{a^i_c}, R^i_{b^i_c}\}_{i \in [\ell]})$

The adversary outputs two pairs of bit vectors $(a_0^i, b_0^i)_{i\in[\ell]}, (a_1^i, b_1^i)_{i\in[\ell]} \in \{0,1\}^{2\ell}$. The game picks a uniformly random challenge $c \in_R \{0,1\}$, samples $\Delta \in_R \{0,1\}^\psi$ and for $i = 1, \ldots, \ell$ it samples $L^i, R^i \in_R \{0,1\}^\psi$, samples $gg^i \leftarrow \mathsf{Yao}(id, L^i, R^i, \Delta)$ and then inputs $(gg^i, L^i_{a^i_c}, R^i_{b^i_c})_{i\in[\ell]}$ to $\mathcal{A}$. Finally $\mathcal{A}$ outputs a bit $d \in \{0,1\}$ and wins if $d = c$. We say that the scheme is IND-CIA if for all PPT $\mathcal{A}$, $\mathcal{A}$ wins the $\mathsf{IND\text{-}CIA}$ game with at most negligible advantage.

Soldering

When a garbled gate gg^1 has the same zero-key (and therefore also one-key) associated to one of its wires, as is associated with one of gg^2's wires, we say that the given wire of gg^1 is *soldered* to the given wire of gg^2. This is a useful concept when we want to build circuits of garbled gates. To see this consider a garbled gate gg^1 with its left input wire soldered to the output of gg^2, and its right input wire soldered to the output of gg^3. This means that if gg^2 and gg^3 has output zero-keys O_0^2 and O_0^3 respectively, then gg^1 has left and right zero-keys $L_0^1 = O_0^2$ and $R_0^1 = O_0^3$. Thus if we evaluate gg^2 and gg^3 on some input and obtain output keys O_a^2 and O_b^3 we can use this to further evaluate gg^1 on these outputs. The resulting output would be some output key $O^1_{a\wedge b}$. S Alternatively notice that if gg^1 has, e.g. left, input zero-key $L_0^1 = O_0^2 \oplus O_0^3$ then

$$O_a^2 \oplus O_b^3 = O_0^2 \oplus O_0^3 \oplus (a \oplus b)\Delta = L_0^1 \oplus (a \oplus b)\Delta = L^1_{a\oplus b} \ .$$

In this case we say that the left input wire of gg^1 is soldered to the XOR of the output of gg^2 and gg^3. This is because by XOR'ing the outputs keys of gg^2 and gg^3 we get the left input key of gg^1 corresponding to XOR of the outputs of gg^2 and gg^3. This is also why we call the garbling *free-XOR*: we do not need to garble XOR gates, since this is handled by the soldering.

In our protocol we will first generate garbled gates where all zero-keys are picked independently, and then in a later stage we will *solder* the wires of the garbled gates to each other to form a garbled circuit. For this purpose, we introduce a function Shift that on input a garbled gate gg, generated as $(gg, O_0) \leftarrow \mathsf{Yao}(id, L_0, R_0, \Delta)$, and three *differences* $dL, dR, dO \in \{0,1\}^\psi$, outputs a new *shifted gate* sgg. The shifted gate sgg is the gate gg modified to have have input zero-keys $(L_0 \oplus dL)$ and $(R_0 \oplus dR)$ and output zero-key $(O_0 \oplus dO)$.

This can be implemented by letting Shift output the concatenation of its inputs i.e., $sgg = (gg, dL, dR, dO)$ and let the evaluation of a shifted gate sgg be defined by:

$$\mathsf{ShiftEval}(sgg, \hat{L}, \hat{R}, \hat{O}) = \mathsf{Eval}(gg, \hat{L} \oplus dL, \hat{R} \oplus dR) \oplus dO$$

where for all K we define $\perp \oplus K = \perp$. It is clear that a shifted gate is correct (with respect to the shifted zero-keys) iff a standard gate is correct, and clearly shifting a gate does not threaten its security property. A shifted gate can be shifted again: The Shift function will just update the values dL, dR, dO accordingly.

Consider two garbled gates $(gg^1, O_0^1) \leftarrow \mathsf{Yao}(1, L_0^1, R_0^1, \Delta)$ and $(gg^2, O_0^2) \leftarrow \mathsf{Yao}(2, L_0^2, R_0^2, \Delta)$. The shifted gate $sgg = \mathsf{Shift}(gg^2, (O_0^1 \oplus L_0^2), 0, 0)$ then becomes a garbled gate with left zero-key $L_0^2 \oplus (O_0^1 \oplus L_0^2) = O_0^1$. I.e. the output wire of gg^1 is now soldered to the left input wire of sgg. Similarly we can use the Shift function to solder a wire to the XOR of some other garbled gates. Namely, consider third gate $(gg^3, O_0^3) \leftarrow \mathsf{Yao}(3, L_0^3, R_0^3, \Delta)$ and $sgg = \mathsf{Shift}(gg^3, ((O_0^1 \oplus O_0^2) \oplus L_0^3), 0, 0)$. The shifted gate ssg then becomes a garbled gate with left zero-key $L_0^3 \oplus ((O_0^1 \oplus O_0^2) \oplus L_0^3) = O_0^1 \oplus O_0^2$. I.e. the left input wire of sgg is soldered to the XOR of gg^1 and gg^2's output wires.

6.2.2 Generic Commitment with Homomorphic Opening

To securely implement the soldering described above, we can not simply have (potentially malicious) A send the differences needed to shift the gates. Instead we will have A give homomorphic commitments to all zero-keys of each gate, and then have her open the differences of the committed keys. Therefore we need a homomorphic commitment scheme. In Figure 6.2 we restate the ideal functionality for the homomorphic commitments we implemented in Chapter 5. As we are now going to use this functionality to implement $\mathcal{F}_{\text{SFE}}$ we will briefly recap the features of this functionality.

The functionality allows A to commit to messages and to later reveal those messages. In addition the functionality allows to reveal the XOR of two or more committed messages to B (without revealing any extra information about the original committed messages).

The functionality is "insecure", in the sense that A can choose a set of up to σ wildcard commitments where she can change her mind about the committed value at opening time. However, openings need to be consistent. More specifically, the $\mathcal{F}_{\text{WCOM}}$ functionality stores a system of linear equations. Initially these equations simply specify that non-wildcard commitments must be opened to the value, they were commitments to. Every time A performs an opening involving wildcard commitments this defines a new linear equation, which is stored in the ideal functionality. For an opening of a wildcard commitment to be successful the set of linear equations stored in the ideal functionality must be consistent.

If the set of equations stored in the ideal functionality constricts the opening of a commitment in such a way that it can only be opened to *one* value, we say that the commitment is *fixed* to that value. Note, that all non-wildcard commitments are fixed, and a fixed wildcard commitment can essentially be viewed as a non-wildcard commitment.

Apart from the regular openings the functionality allows to open (the XOR of) committed messages in two alternative ways: In an *Oblivious-Opening*, B can choose between two sets of committed messages and learn the XOR of the messages in one of them. In an *Or-Opening* we allow A to open the XOR of one out of two sets of committed messages without revealing which one. For technical reasons there can only be a total of ℓ Or-Openings and all Or-Openings must be done before the first Oblivious-Opening. Also, note that there is a build-in selective failure attack in the Oblivious-Opening. However, this is not

Init

On input (ID, W) from the adversary, with $|ID| = \Gamma$, $|W| \leq \sigma$ and $W \subset ID$, output ID to both parties and let $\mathcal{J} = \emptyset$. If A is honest, then $W = \emptyset$.

Commit

On input $(\texttt{commit}, j \in ID, m_j)$ with $m_j \in \{0,1\}^\sigma$ from A, and where no value of the form $(j, \cdot)$ is stored, store (j, m_j). If $j \in ID \setminus W$, add $J = \{j\}$ to $\mathcal{J}$ and associate with J the equation $\mathtt{X}_j = m_j$. Then output $(\texttt{commit}, j)$ to B.

Open

On input $(\texttt{open}, J \subset ID)$ from A, where for all $j \in J$ a pair (j, m_j) is stored do the following:

- If A is honest, output $(\texttt{open}, J, \oplus_{j \in J} m_j)$ to B.
- If A is corrupted wait for A to input $(\texttt{corrupt-open}, J, m_J)$. Then add J to $\mathcal{J}$, associate the equation $\oplus_{j \in J} \mathtt{X}_j = m_J$ to J, and check that the equation system $\{\oplus_{j \in J} \mathtt{X}_j = m_J\}_{J \in \mathcal{J}}$ has a solution. If so, output $(\texttt{open}, J, m_J)$ to B. Otherwise, output `Alice cheats` to B and terminate.

Oblivious Opening

On input $(\texttt{OT-choose}, otid, b)$ with $b \in \{0,1\}$ from B output $(\texttt{OT-choose}, otid)$ to A. On input $(\texttt{OT-open}, otid, J_0, J_1)$ from A with $J_0, J_1 \subset ID$ where for all $j \in J_0, J_1$ a pair (j, m_j) is stored and $(\texttt{OT-choose}, otid, *)$ was input before by B do the following:

- If A is honest, output $(\texttt{OT-open}, otid, J_b, \oplus_{j \in J_b} m_j)$ to B (Note that B does not learn the set of ids J_{1-b}).
- If A is corrupted, wait for A to input $(\texttt{guess}, g)$ with $g \in \{0,1,\perp\}$. If $g \in \{0,1\}$ and $g \neq b$ output `Alice cheats` to B and terminate. Otherwise, proceed to wait for A to input $(\texttt{corrupt-open}, J_0, J_1, m_{J_0}, m_{J_1})$. Add J_b to $\mathcal{J}$ and associate the equation $\oplus_{j \in J_b} \mathtt{X}_j = m_{J_b}$ to J_b. Check that the equation system still has a solution as described above. If so, output $(\texttt{OT-open}, J_b, m_{J_b})$ to B. Otherwise output `Alice cheats` to B.

OR Open

On input $(\texttt{OR-open}, J_0, J_1, a)$ from A, with $J_0, J_1 \subset ID, a \in \{0,1\}$ where for all $j \in J_0, J_1$ a pair (j, m_j) is stored do the following:

- If A is honest, output $(\texttt{OR-open}, J_0, J_1, \oplus_{j \in J_a} m_j)$ to B.
- If A is corrupted, and if $J_a \cap W \neq \emptyset$, wait for corrupt A to input $(\texttt{corrupt-open}, J_a, m_{J_a})$, add J_a to $\mathcal{J}$ and associate $\oplus_{j \in J_a} \mathtt{X}_j = m_{J_a}$ to J_a. Check if the equation system still has a solution as described above. If so, output $(\texttt{OR-open}, J_0, J_1, m_{J_a})$ to B. Otherwise output `Alice cheats` to B.

There can at most be ℓ such openings, and they must all occur before the first Oblivious-Opening.

Figure 6.2: The $\mathcal{F}_{\text{WCOM}}^{\sigma,\Gamma,\ell}$ functionality for Γ commitments with ℓ Or-Openings.

a problem as we will only use this type of opening to handle B's input where, as discussed above, the $\mathcal{F}_{\text{SFE}}$ functionality already allows a selective failure attack.

In Cor. 5.1 in Chapter 5 we showed that $\mathcal{F}_{\text{WCOM}}^{\sigma,\Gamma,\omega}$ can be securely implemented for $\Gamma > (c\sigma^2 + \sigma)^2$, for a constant c, in the $(\binom{n}{t}\text{-}\mathcal{F}_{\text{OT}}(4\Gamma + 12\sigma\omega), \mathcal{F}_{\text{OT}})$-hybrid model. The constant c essentially depends on the rate of a code used to in the implementation of of the commitments, for more details see Chapter 5.

Non-Homomorphic Commitments

Additional to the $\mathcal{F}_{\text{WCOM}}$ functionality we are going to use an extractable commitment Com. This commitment is used only once by B to commit to his challenge in the cut-n-choose phase and extraction is needed for simulation (to avoid selective opening issues). Since this commitment does not need to be homomorphic it can be easily implemented in the $\mathcal{F}_{\text{OT}}$-hybrid model.

6.3 Protocol

The protocol π_{LEGO} implementing $\mathcal{F}_{\text{SFE}}$ is described in Figure 6.3. In Section 6.3.1 we first give the notation and conventions we are going to use to describe the protocol, before we in Section 6.3.2 give a more detailed description of the protocol. The security analysis of the protocol follows in Section 6.4.

6.3.1 Notation and Conventions

Plaintext Circuit

We denote by C the original Boolean circuit to be evaluated. We assume C to be composed of AND and XOR gates. The XOR gates are allowed to have unbounded fan-in while the AND gates have fan-in 2. With each AND gate in C we associate a unique label and we let gates be the set of all these labels. A subset $\mathsf{inputGates} \subset \mathsf{gates}$ of size 2ℓ are specially marked as input gates. The AND gates in $\mathsf{inputGates}$ should be given the same bit on both input wires, so that the gate simply computes the identity function. A subset in $\mathsf{Ainputs} \subset \mathsf{inputGates}$ of size ℓ are taken to be A's inputs. The remaining ℓ gates in $\mathsf{Binputs} = \mathsf{inputGates} \setminus \mathsf{Ainputs}$ are B's inputs (for convenience assume that $\mathsf{Binputs} = [\ell]$). A has input bits $(a_1, \ldots, a_\ell)$, while B has input bits $(b_1, \ldots, b_\ell)$.

A subset $\mathsf{outputGates} \subset \mathsf{gates}$ of size ℓ are marked as output gates. The output of these gates are taken to be the output of the circuit. Note that this means that all output gates are AND gates. However, this is without loss of generality: any circuit with one or more XOR gates as output gates can easily be modified to an equivalent circuit with AND gates as output gates by adding at most ℓ AND gates. The ℓ output bits are denoted $(z_1, \ldots, z_\ell)$.

The wiring of the circuit C is described by two functions $\mathsf{lp}, \mathsf{rp} : \mathsf{gates} \setminus \mathsf{inputGates} \to 2^{\mathsf{gates}}$. We call $\mathsf{lp}(j)$ the left parents of j (resp. $\mathsf{rp}(j)$ the right parents of j), and take the left (resp. right) input of j to be the XOR of the output bits of all gates in $\mathsf{lp}(j)$ (resp. $\mathsf{rp}(j)$). Thus, the XOR gates of C are implicitly defined by the lp and rp functions. We assume that C does not have loops and it is ordered i.e., $\max(\mathsf{lp}(j)) < j$ and $\max(\mathsf{rp}(j)) < j$.

Garbled Circuit

Let $\Gamma = 2\rho s$ for $s = |\mathsf{gates}|$ and some replication factor $\rho \in \mathbb{N}$. For our protocol A will construct Γ garbled gates. She constructs twice as many garbled gates as is needed to build the garbled circuit, because half the gates are going to be checked during the cut-n-choose phase. We choose to check exactly half for the

sake of presentation but, as in [SS11], this could be changed to any fraction in order to optimize concrete efficiency.

Let $\mathcal{B}$ be the family of ρ-to-1, ρ-wise independent functions from a set $U \subset [\Gamma]$ of size ρs to gates. For a function $\mathsf{BucketOf} \in \mathcal{B}$ let Bucket be the function that, for all $j \in \mathsf{gates}$ outputs the set $\{i \in U | \mathsf{BucketOf}(i) = j\}$. Let $\mathsf{BucketHead}(j)$ be the function that returns the "first" (in lexicographic order) element of $\mathsf{Bucket}(j)$.

There are $3\Gamma + 1$ keys in the protocol, because every constructed AND gate has a left, right and output key and in addition there is a global difference Δ. The key index is written as a superscript while subscripts are in $\{0, 1\}$ and describe the value carried by the key i.e., $K_b^i = K^i \oplus b\Delta$. Let id be a function that on input a key $K_0^j \in \{0,1\}^\psi$ returns a unique label for that key. We will sometimes abuse notation and write $\mathsf{id}(K_1^j)$ to denote the set $\{\mathsf{id}(K_0^j), \mathsf{id}(\Delta)\}$. This will simplify the notation when using the $\mathcal{F}_{\text{WCOM}}$ functionality.

6.3.2 Protocol Description

The protocol π_{LEGO} in Figure 6.3 progresses in six phases: **Setup**, **Garbling**, **Cut-n-Choose**, **Soldering**, **Input** and **Evaluation**. Here we describe these phases one by one.

During **Setup**, A and B initialize a $\mathcal{F}_{\text{WCOM}}^{\sigma,3\Gamma+1,\ell}$ functionality, and for the remainder of the protocol if $\mathcal{F}_{\text{WCOM}}$ outputs `Alice cheats` , B will abort the protocol. Then A samples the global difference Δ and commits to it using $\mathcal{F}_{\text{WCOM}}$. B samples his challenge for the cut-n-choose phase and the $\mathsf{BucketOf}$ function as described above, and commits to both using the extractable commitment Com. B also "commits" to his input using the `OT-choose` command of the $\mathcal{F}_{\text{WCOM}}$ functionality. These commitments of B's are needed to avoid selective opening issues in the cut-n-choose phase and reduce the security of the protocol to the IND-CIA game.

In **Garbling**, A constructs the candidate garbled gates $(gg^i)_{i\in[\Gamma]}$ and commits to the input/output zero-keys of each garbled gate using $\mathcal{F}_{\text{WCOM}}$.

In **Cut-n-Choose**, B reveals his challenge. The challenge consists of a set of indices $T \subset [\Gamma]$ of size $s\rho$ and a sequence of bits $(u_i, v_i)_{i\in T}$, indicating that B wants to test garbled gate gg^i on input (u_i, v_i). A opens the corresponding input and output keys for the test gates, allowing B to check for correctness. Note that B only test one set of inputs for each gate – otherwise he will learn Δ.

In the remainder of the protocol the garbled gates that are not checked in **Cut-n-Choose**, those with indices in $U = [\Gamma] \setminus T$, are used to build a garbled circuit according to the following fault tolerant circuit design: With each gate $j \in \mathsf{gates}$ we associate a *bucket* of ρ AND gates. To evaluate gate j we will evaluate each gate in the bucket of j on the inputs given to j. If more than $\rho/2$ of the gates in the bucket agree on their output bit, we take this bit to be the output of j (otherwise the output is $\perp$). Clearly if there are at more than $\rho/2$ non-faulty gates in each bucket the output obtained in this way is correct.

To build such a garbled circuit the gates that were not checked $(gg^i)_{i\in U}$ are assigned into buckets using the $\mathsf{BucketOf}$ function. Then B uses the Shift

function as described in Section 6.2.1 to solder the wires of the garbled gates. Note that since A may be malicious we cannot simply have her sent the XOR's of zero-keys that B needs for soldering. Instead A reveals the XOR's by opening the corresponding commitments to the zero-keys.

In this way the garbled circuit is constructed in **Soldering** in three different soldering steps: For all $j \in \mathsf{gates}$ **Horizontal Soldering** solders all wires of all gates in $(gg^i)_{i\in\mathsf{Bucket}(j)}$ to the corresponding wires of $gg^{\mathsf{BucketHead}(j)}$. This allows to evaluate all the gates in the same bucket on the same input keys and get the same output keys. I.e. if A is honest, after the horizontal soldering all the gates in one bucket have exactly the same keys. For all $j \in \mathsf{gates}$ **Vertical Soldering** solders the left input wire of $gg^{\mathsf{BucketHead}(j)}$ to the XOR of the output wires of $(gg^{\mathsf{BucketHead}(i)})_{i\in lp(j)}$, and the right input wire of $gg^{\mathsf{BucketHead}(j)}$ to the XOR of the output wires of $(gg^{\mathsf{BucketHead}(i)})_{i\in rp(j)}$. Note that since **Horizontal Soldering** made all garbled gates in a bucket have the same input keys, this essentially means soldering *all* the gates in the bucket to the output wires of gates in $(\mathsf{Bucket}(i))_{i\in\mathsf{lp}(j)}$ and $(\mathsf{Bucket}(i))_{i\in\mathsf{rp}(j)}$. I.e. vertical soldering is "functional", in the sense that it make sure that the circuit computes the right circuit, C. For all $j \in \mathsf{inputGates}$ **Input Soldering** simply solders the left and right input wire of garbled gates in $\mathsf{Bucket}(j)$ to each other. This means that the gates in inputGates simply compute the identity function.

In **Input**, for all $j \in \mathsf{Ainputs}$ A uses the Or-Opening of $\mathcal{F}_{\mathrm{WCOM}}$ to open the input key to the garbled gates in $\mathsf{Bucket}(j)$ corresponding to her input bit. For all $j \in \mathsf{Binputs}$ B also learns the input key to the garbled gates in $\mathsf{Bucket}(j)$ corresponding to his input bit, using the Oblivious-Opening.

Given the initial input keys in **Evaluation** B evaluates each bucket of garbled gates in the following way: He evaluates each gate in the bucket on the left and right input keys for that bucket. If a key appears more than $\rho/2$ times as the output key of the garbled gates in the bucket, he takes this to be the output key of the bucket. If no such key exists B aborts. Note that by the way we soldered the garbled circuit, this corresponds exactly to the fault tolerant circuit we described above. Finally B provides A with the output keys. Knowing Δ, A can decipher the output keys and obtain the output values.

In Section 6.4 we prove the following theorem.

Theorem 6.1. *Let σ be the security parameter, $\rho \geq \frac{7\sigma+12}{2(\log(s)-7)}$ and $\Gamma = 2\rho s$. If $(\mathsf{Yao}, \mathsf{Eval})$ is an IND-CIA secure* Yao free-xor garbling scheme *then the protocol π_{LEGO} in Figure 6.3 UC, active, static securely implements $\mathcal{F}_{\mathrm{SFE}}$ in the $(\mathcal{F}_{\mathrm{WCOM}}^{\sigma,3\Gamma+1,\ell})$-hybrid model.*

6.4 Analysis

In this section we split the proof of Theorem 6.1 in two, proving first in Section 6.4.1 security against a corrupted B and in Section 6.4.2 security against corrupted A. Combining the two lemmas we get Theorem 6.1.

6.4.1 Corrupted B

B does not receive any output nor has any real way of cheating in the protocol (in the output phase, if B changes the output key in a way that makes A accept, then he must have guessed Δ, thus breaking the IND-CIA game). Essentially, we only need to argue that his view does not leak any information, thanks to the IND-CIA security of the garbling scheme. Note that in the protocol B starts by committing to his input and challenge for the cut-n-choose phase. This allows the simulator S to extract all this information at the beginning of the simulation (and provide input on behalf of corrupted B to the ideal functionality). Then we reduce the security of the protocol to the IND-CIA security of the garbling scheme: the simulator knows in fact T and U before it sends the gates to B, therefore S will place honestly constructed gates in T (for which it knows the openings and therefore can easily simulate the cut-n-choose test – remember that the simulator fully controls $\mathcal{F}_{\text{WCOM}}$) and the challenge garbled gates from the IND-CIA game in U: that is, the simulator produces a view such that distinguishing between a real and a simulated execution is equivalent to winning the IND-CIA game.

Lemma 6.1. *π_{LEGO} is a secure implementation of $\mathcal{F}_{\text{SFE}}$ against a malicious* B^*.

Proof. We present a simulator S that given access to $\mathcal{F}_{\text{SFE}}$ simulates the real world view of the environment Z when Z corrupts B^*.

At the beginning of the simulation Z inputs (a, c_{A}) and (b, c_{B}) to A and S respectively, and A inputs (a, c_{A}) to $\mathcal{F}_{\text{SFE}}$. The simulator receives $c_a = (\ell, C)$ from $\mathcal{F}_{\text{SFE}}$ and then runs π_{LEGO} completely as an honest A with input $(0^\ell, c_{\mathsf{A}})$ except that the simulator fully controls the $\mathcal{F}_{\text{WCOM}}$ functionality. Thus S reads the input b^* used by B^* as it is given to the $\mathcal{F}_{\text{WCOM}}$ functionality in step **Setup**-6, and S can extract the challenges chosen by B in step **Setup**-5. To conclude the simulation, if B^* does not cause the protocol to abort, S inputs (c_B, b^*) to $\mathcal{F}_{\text{SFE}}$ on behalf of B^* and $\mathcal{F}_{\text{SFE}}$ outputs $C(a, b^*)$ to A. If B^* causes S to abort, S makes $\mathcal{F}_{\text{SFE}}$ abort.

To argue indistinguishability of the Z's view in the real and ideal world we reduce to the security of garbling scheme (Yao, Eval). Thus we consider an adversary $\mathcal{A}$ in the IND-CIA game as defined in Def. 6.1 that makes use of Z and B^*. We will construct $\mathcal{A}$ so that, depending on the value of c chosen by the IND-CIA game, $\mathcal{A}$ produces a view for Z that is either indistinguishable from the real world view or the ideal world simulation of S.

$\mathcal{A}$ first runs Z to get the inputs (a, c_a) for A. When B^* in **Setup**-5 and -6 inputs T, $(u_i, v_i)_{i \in T}$, BucketOf and b^* $\mathcal{A}$ reads these values.

Now $\mathcal{A}$ uses the IND-CIA game to generate the garbled gates to hand to B^*. $\mathcal{A}$ constructs its two strings to IND-CIA so that $\mathcal{A}$ receives keys to evaluate the garbled circuit, as specified by T and BucketOf, on inputs (a, b^*) for $c = 0$, and on inputs $(0^\ell, b^*)$ for $c = 1$. Additionally $\mathcal{A}$ will construct the strings so that for either value of c, $\mathcal{A}$ receives the keys needed to evaluate the gates in the cut-n-choose challenge.

To this end $\mathcal{A}$ evaluates $C(a, b^*)$ and for each gate $j \in \mathsf{gates}$ records the left and right inputs l_0^j and r_0^j respectively. Similarly $\mathcal{A}$ evaluates $C(0^\ell, b^*)$ and for each gate $j \in \mathsf{gates}$ records the left and right inputs l_1^j and r_1^j. Then $\mathcal{A}$ outputs the strings $(\hat{a}_0^1, \ldots, \hat{a}_0^\Gamma, \hat{b}_0^1, \ldots, \hat{b}_0^\Gamma)$ and $(\hat{a}_1^1, \ldots, \hat{a}_1^\Gamma, \hat{b}_1^1, \ldots, \hat{b}_1^\Gamma)$ where

- For each $i \in T$ $\mathcal{A}$ lets $(\hat{a}_0^i, \hat{b}_0^i) = (\hat{a}_1^i, \hat{b}_1^i) = (u_i, v_i)$.
- For each $j \in \mathsf{gates}$ $\mathcal{A}$ lets $\hat{a}_0^i = l_0^j$ and $\hat{a}_1^i = l_1^j$ for all $i \in \mathsf{Bucket}(j)$.
- For each $j \in \mathsf{gates}$ $\mathcal{A}$ lets $\hat{b}_0^i = r_0^j$ and $\hat{b}_1^i = r_1^j$ for all $i \in \mathsf{Bucket}(j)$.

Given these strings the IND-CIA game inputs $(gg^i, L^i_{\hat{a}_c^i}, R^i_{\hat{b}_c^i})_{i\in[\Gamma]}$ to $\mathcal{A}$, and $\mathcal{A}$ hands $(gg^i)_{i\in[\Gamma]}$ to B^*.

$\mathcal{A}$ then computes the output keys $O^i \leftarrow \mathsf{Eval}(gg^i, L^i_{\hat{a}_c^i}, R^i_{\hat{b}_c^i})$ for each $i \in [\Gamma]$. Notice that the keys $(L^i_{\hat{a}_c^i}, R^i_{\hat{b}_c^i}, O^i)_{i\in[\Gamma]}$ are enough for $\mathcal{A}$ to compute the correct value of all the strings she needs to open in **Cut-n-Choose** and **Soldering**: For **Cut-n-Choose** $\mathcal{A}$ knows $L^i_{u_i} = L^i_{\hat{a}_c^i}$, $R^i_{v_i} = R^i_{\hat{b}_c^i}$ and $O^i = O^i_{u_i \wedge v_i}$. For **Soldering** consider the difference dL^h corresponding to a gate $j \in \mathsf{gates}$ with $h = \mathsf{BucketHead}(j)$. $\mathcal{A}$ can compute dL^h as

$$\begin{aligned} dL^h &= L^h_{\hat{a}_c^h} \oplus O^{\mathsf{BucketHead}(\mathsf{lp}(j))} \\ &= (L^h_0 \oplus \hat{a}_c^h \Delta) \oplus (O_0^{\mathsf{BucketHead}(\mathsf{lp}(j))} \oplus \hat{a}_c^h \Delta) \\ &= L^h_0 \oplus O_0^{\mathsf{BucketHead}(\mathsf{lp}(j))} \end{aligned}$$

where the second equality is by the correctness of $(\mathsf{Yao}, \mathsf{Eval})$ and definition of $\hat{a}_c^h$ and $\hat{b}_c^h$. Similarly $\mathcal{A}$ can compute all the other differences needed in **Soldering**.

Thus $\mathcal{A}$ can simulate **Cut-n-Choose** and **Soldering** as it fully controls the $\mathcal{F}_{\mathrm{WCOM}}$ functionality.

Similarly $\mathcal{A}$ can simulate **Input** by simply sending the input keys given by the IND-CIA game to B^* on behalf of the $\mathcal{F}_{\mathrm{WCOM}}$ functionality.

In **Evaluation** $\mathcal{A}$ receives $\hat{O}^j$ for each $j \in \mathsf{outputGates}$ from B^*. Now since $\mathcal{A}$ has all the same keys for the garbled circuit as B^*, she can compute the keys O^j that an honest B would have sent. Thus A aborts if $\hat{O}^j \neq O^j$. Otherwise, $\mathcal{A}$ outputs $C(a, b^*)$ to Z on behalf of A. $\mathcal{A}$ then outputs whatever Z outputs.

Denote by F the event that B^* outputs a $\hat{O}^j \neq O^j$ so that $\hat{O}^j \oplus O^j = \Delta$, and assume F does not occur. Then by definition of the IND-CIA game and the way $\mathcal{A}$ constructs the strings $(\hat{a}_0^1, \ldots, \hat{a}_0^\Gamma, \hat{b}_0^1, \ldots, \hat{b}_0^\Gamma)$ and $(\hat{a}_1^1, \ldots, \hat{a}_1^\Gamma, \hat{b}_1^1, \ldots, \hat{b}_1^\Gamma)$ it is easy to verify that when $c = 0$ the view produced by $\mathcal{A}$ towards Z is perfectly indistinguishable to the real world view, while for $c = 1$ the view is perfectly indistinguishable from the ideal world simulation of S. Thus any advantage of Z in distinguishing the real from ideal world directly translates into advantage of $\mathcal{A}$ in the IND-CIA game. Thus assuming F does not occur and $(\mathsf{Yao}, \mathsf{Eval})$ is secure the ideal and real worlds are indistinguishable.

Now assume F does occur. In this case $\mathcal{A}$ aborts the protocol where as S and honest A does not. To handle this we argue that F only occurs with negligible

probability. In fact, if F occurs and B^* outputs a $\hat{O}^j \neq O^j$ then $\mathcal{A}$ can use $\Delta' = \hat{O}^j \oplus O^j$ to get all keys for some garbled gate and therefore distinguish. By assumption that F does occur this allows $\mathcal{A}$ to win the IND-CIA game with noticeable probability. □

6.4.2 Corrupted A

Essentially, the proof of security boils down to proving correctness. By the design of the garbled circuit correctness follows when if there is more than $\rho/2$ correct gates in each of the buckets.

The LEGO approach ensures that if A passes the cut-n-choose test, then with overwhelming probability there are at most $O(\sigma)$ faulty gates left in U. Those faulty gates are then randomly assigned into buckets, and this means that with overwhelming probability each bucket will have a majority of correct gates.

However, as opposed to [NO09] where all commitments were binding, here we have also σ wildcard commitments to deal with. This is in problematic, as wildcard commitments can be opened to anything, and we need make sure that this does not break correctness.

To be more specific we say that a garbled gate gg^i is *faulty* if the commitments to its input and output zero-keys are fixed to values L_0^i, R_0^i and O_0^i respectively, and there exists some $a, b \in \{0,1\}$ so that $\mathsf{Eval}(gg^i, L_a^i, R_b^i)$ does not output $O_{a\wedge b}^i$ with overwhelming probability. If a gate gg^i has a wire where the commitment to the associated zero-key is not fixed, then we say that this wire is faulty, and gg^i has *faulty wiring*. We say that gg^i is *fault free* if it is neither faulty nor has faulty wiring. If a garbled gate gg^i is faulty, fault free or has faulty wiring, we say the same of any shifted gate sgg^i resulting from shifting gg^i.

Gates gg^i with faulty wiring are problematic for the cut-n-choose test: If $i \in T$ A can choose to let gg^i act as a fault free gate by opening the wildcard commitments consistently with the actual zero-keys used to generate gg^i. On the other hand, if $i \in U$ A can make sgg^i faulty by opening the commitment inconsistently in **Soldering** [1].

In Lemma 6.2 we show that, with overwhelming probability, there will be a majority of fault free gates in $(gg^i)_{i\in\mathsf{Bucket}(j)}$ for all $j \in \mathsf{gates}$. It is easy to verify that this means that after **Horizontal Soldering** all commitments to zero-keys are fixed. I.e. the commitment to the zero-key of a faulty wire will be fixed to open as one specific value. If this value is not consistent with the zero-keys used to generate the associated garbled gate, then that gate becomes faulty.

Note however, that for all $j \in \mathsf{gates}$ all fault free shifted gates $(sgg^i)_{i\in\mathsf{Bucket}(j)}$ resulting from **Horizontal Soldering** will have identical input and output keys, as required of the garbled circuit, even if some gates in $(gg^i)_{\mathsf{Bucket}(j)}$ had faulty wiring. I.e. the effect of a garbled gate gg^i having faulty wiring is *at worst* that shifted gate sgg^i after **Soldering** is faulty. Since we use a $\mathcal{F}_{\text{WCOM}}$

[1] By *inconsistently* we mean inconsistent with the actual keys used for gg^i, not inconsistent with the equations stored in $\mathcal{F}_{\text{WCOM}}$

functionality with at most σ wildcard commitments we still have at most $O(\sigma)$ faulty gates in **Evaluation**. Since these faulty gates are placed in random buckets we can still guarantee correctness with overwhelming probability.

Note that the faulty wires are also why we replicate the gates on the input layer, to not let A change her or B's input by using the wildcard commitments.

Before we show security for corrupted A, we show in Lemma 6.2 that if the protocol does not abort in *Cut-n-Choose* then for the remainder of the protocol there will be more than $\rho/2$ fault free garbled gates in each of the buckets with overwhelming probability. We note that in [Orl11] a similar lemma is proved, however, there they did not have the wildcard commitments to worry about.

Lemma 6.2. *Consider the protocol π_{LEGO} with honest B. Let σ be the security parameter, $\rho \geq \frac{7\sigma+12}{2(\log(s)-7)}$, and assume that A in **Garbling** prepares f faulty gates and v gates with faulty wiring. Let*

- *E_1 be the event that the protocol does not abort in **Cut-n-Choose**.*
- *E_2 be the event that, before **Soldering**, there exists a $j \in \mathsf{gates}$ so that $(gg^i)_{i\in\mathsf{Bucket}(j)}$ does not have more than $\rho/2$ fault free gates.*

Then $\Pr(E_1 \wedge E_2)$ is at most

$$s\left(\frac{\sigma+f}{s\rho}\right)^{\lceil\rho/2\rceil} 2^{\rho-(f/6+1)} \leq 2^{-\sigma} \ .$$

Proof. Let $F, V \subset [\Gamma]$ be the sets of indices of all garbled gates that are faulty or have faulty wiring respectively and let $G = F \cup V$. Note that by the definition of faulty gates and faulty wiring $F \cap V = \emptyset$. Let $b = |G| = f+v$, $r_i = |\mathsf{Bucket}(i) \cup G|$ and $R = |\{i \in [s] | r_i > \rho/2\}$ (Here we assume ρ to be odd).

Then for each garbled gate in $\{gg^i\}_{i\in F}$ we have that $i \in T$ with probability $1/2$, and that if $i \in T$ then gg^i will be detected as being faulty with probability at least $1/4$. I.e. we have

$$\Pr(E_1) \leq \left(\frac{7}{8}\right)^f = \left(\frac{8}{7}\right)^{-f} \leq 2^{-f/6} \ .$$

To bound $\Pr(E_2) \leq \Pr(R > 0)$, first notice that $\Pr(r_i > \rho/2)$ is the same for any $i \in [s]$. Thus by union bound we have

$$\Pr(R > 0) \leq s \cdot \Pr(r_1 > \rho/2) \ . \tag{6.1}$$

The event $r_1 > \rho/2$ we can describe in terms of the following experiment: from a collection of ρs balls were b balls are red and $\rho s - b$ balls are green pick at random ρ balls. The probability that we pick t red balls in this way is exactly $\Pr(r_1 = t)$. I.e. r_1 follows a hyper geometric distribution and we have

$$\Pr(r_1 = t) = \binom{b}{t}\binom{s\rho-b}{\rho-t}\binom{s\rho}{\rho}^{-1} \ .$$

Note that we can assume $b \geq t$ or this probability is clearly 0. Writing out the binomial coefficients we get

$$\Pr(r_1 = t) = \prod_{i=0}^{t-1} \frac{b-i}{t-i} \prod_{i=0}^{\rho-t-1} \frac{s\rho-b-i}{\rho-t-i} \prod_{i=0}^{\rho-1} \frac{\rho-i}{s\rho-i} \ . \tag{6.2}$$

Assume $\rho \geq t > \rho/2$ and focus on the last two products of (6.2). Then we have

$$\begin{aligned}\prod_{i=0}^{\rho-t-1} \frac{s\rho-b-i}{\rho-t-i} \prod_{i=0}^{\rho-1} \frac{\rho-i}{s\rho-i} &= \prod_{i=0}^{\rho-t-1} \frac{s\rho-b-i}{\rho-t-i} \prod_{i=0}^{\rho-t-1} \frac{\rho-t-i}{s\rho-t-i} \prod_{i=0}^{t-1} \frac{\rho-i}{s\rho-i} \\ &\leq \prod_{i=0}^{t-1} \frac{\rho-i}{s\rho-i} \ ,\end{aligned}$$

where the inequality follows from $b \geq t$. If we plug this into (6.2) we get

$$\begin{aligned}\Pr(r_1 = t) &\leq \prod_{i=0}^{t-1} \frac{b-i}{t-i} \prod_{i=0}^{t-1} \frac{\rho-i}{s\rho-i} \\ &= \prod_{i=0}^{t-1} \frac{\rho-i}{t-i} \prod_{i=0}^{t-1} \frac{b-i}{s\rho-i} \\ &= \binom{\rho}{t} \prod_{i=0}^{t-1} \frac{b-i}{s\rho-i} \\ &\leq \binom{\rho}{t} \left(\frac{b}{s\rho}\right)^t \ ,\end{aligned}$$

Plugging this into (6.1) we have

$$\begin{aligned}\Pr(R > 0) &\leq s \Pr(r_1 > \rho/2) \\ &\leq s \sum_{t=\lceil \rho/2 \rceil}^{\rho} \binom{\rho}{t} \left(\frac{b}{s\rho}\right)^t \\ &\leq s \left(\frac{b}{s\rho}\right)^{\lceil \rho/2 \rceil} \sum_{t=\lceil \rho/2 \rceil}^{\rho} \binom{\rho}{t} \\ &= s \left(\frac{b}{s\rho}\right)^{\lceil \rho/2 \rceil} 2^{\rho-1} \ .\end{aligned}$$

Now since $v \leq \sigma$ by definition of the $\mathcal{F}_{\text{WCOM}}$ functionality used in π_{LEGO}, we have as stated in the lemma

$$\Pr(E_1 \wedge E_2) \leq s \left(\frac{f+\sigma}{s\rho}\right)^{\lceil \rho/2 \rceil} 2^{\rho-1} 2^{-f/6} \ .$$

It is easy to verify that this expression is maximized for $f = 6\lceil\rho/2\rceil/\ln(2) - \sigma$. Thus we have

$$\begin{aligned}
\Pr(E_1 \wedge E_2) &\leq s\left(\frac{6\lceil\rho/2\rceil/\ln(2)}{s\rho}\right)^{\lceil\rho/2\rceil} 2^{\rho-1-\lceil\rho/2\rceil/\ln(2)+\sigma/6} \\
&\leq s\left(\frac{5}{s}\right)^{\lceil\rho/2\rceil} 2^{\rho-2\rho/3+\sigma/6} \\
&= 2^{-\log(s)(\rho-1)/2+\log(5)(\rho+1)/2+\rho/3+\sigma/6} \\
&\leq 2^{-\log(s)\rho/3+7\rho/3+2+\sigma/6} \\
&\leq 2^{\rho(7-\log(s))/3+2+\sigma/6},
\end{aligned}$$

where the third inequality is from $(\rho-1)/2 \geq \rho/3$ which follows from the assumption on ρ. From $\rho \geq \frac{7\sigma+12}{2(\log(s)-7)}$ it follows that $\Pr(E_1 \wedge E_2) \leq 2^{-\sigma}$. □

Lemma 6.3. *π_{LEGO} is a secure implementation of $\mathcal{F}_{\text{SFE}}$ against a malicious* A.

Proof. We present a simulator S that given access to $\mathcal{F}_{\text{SFE}}$ simulates the real world view of the environment Z when Z corrupts A^*.

At the beginning of the simulation Z inputs (b, c_{B}) to B who inputs it to $\mathcal{F}_{\text{SFE}}$. So S receives $c_{\mathsf{B}} = (\ell, C)$ from $\mathcal{F}_{\text{SFE}}$ and then runs π_{LEGO} as an honest B with input $(0^\ell, c_{\mathsf{B}})$ towards A^*.

In **Input**-2 S can not immediately simulate the $\mathcal{F}_{\text{WCOM}}$ functionality as it does not know the input b of the honest B. I.e. it does not know if the $\mathcal{F}_{\text{WCOM}}$ would output `Alice cheats` making the real world protocol abort. However, S can check for each $j \in \mathsf{Binputs}$ if there is a value of $b_j \in \{0,1\}$ that *would* make $\mathcal{F}_{\text{WCOM}}$ output `Alice cheats` . For each j where such a value of b_j exists denote that value $\hat{b}_j$ (if more than one such value exists S can safely abort the protocol) and let B be set of all such j's . Then S sets $\beta_j = 1 - b'_j$ and inputs $(j, \beta_j)_{j\in B}$ to $\mathcal{F}_{\text{SFE}}$. Note, that this perfectly simulates the selective failure attack that A^* might do.

Since S fully controls $\mathcal{F}_{\text{WCOM}}$ through out the protocol, S can extract any information A^* inputs to $\mathcal{F}_{\text{WCOM}}$. This includes all committed keys, the indices of wildcard commitments W and A^*'s input a^*.

Also, S can extract the value Δ from the commitment done in **Setup**-2. The only problem is that if the commitment to Δ is a wildcard commitment, S cannot be sure that A^* will use this value of Δ later in the protocol. However, even if the commitment to Δ is a wildcard commitment it will become fixed as soon as it is opened XOR a non-wildcard commitment. This happens in **Cut-n-Choose** if for some $i \in T$ $u_i = 1$ and $\mathsf{id}(L^i) \notin W$ (or $v_i = 1$ and $\mathsf{id}(R^i) \notin W$ respectively). Say that this does occurs for some $i \in T$ and A^* opens the XOR of the two commitments to K, then since S can extract the value L^i it can also compute $\Delta = L^i \oplus K$ which is the value that the commitment to Δ is now fixed to open to by $\mathcal{F}_{\text{WCOM}}$.

The probability that Δ does not become fixed in this way some $i \in T$ is negligible. To see this consider that there is at most σ wildcard commitments but $|T| = \rho s$, thus the since the challenges u_i and v_i are sampled uniformly at

random, the probability that Δ is not fixed is at most $2^{\sigma - s\rho}$. By assumption that $s \geq \sigma^4$ this probability is negligible. I.e. regardless of whether or not A^* commits to Δ using a wildcard commitment, S learns Δ with overwhelming probability.

If A^* did not make the protocol abort, S inputs a^* to $\mathcal{F}_{\text{SFE}}$ and learns the output z.

By Lemma 6.2 and the discussion above we have that, with overwhelming probability, all buckets have a majority of fault free shifted gates that have the same input and output keys. S can now, for all $j \in \mathsf{outputGates}$, compute keys corresponding to the actual outputs, i.e., $O^j_{z_j} = O^j_0 \oplus z_j\Delta$, where O^j_0 is the output zero-key of the fault free gates in $\mathsf{Bucket}(j)$ (which where extracted earlier).

Now we argue that the view of A^* is statistically indistinguishable (in the $\mathcal{F}_{\text{WCOM}}$-hybrid model) when playing with a real B and when playing with the simulator S. This is by construction: note that in the protocol the only information that travels from B to A^* is

1. in step **Cut-n-Choose**-1, when B opens the commitment Com and the simulator does this as an honest B would do.

2. B's reaction to the selective failure attack A^* may use in **Input**-2, but as we have argued above, this is simulated perfectly.

3. In **Evaluate**-3, when B reveals the output keys to A.

So as long as the simulator S sends the correct output keys to A^*, the views are identical.

It follows from the construction that the simulator S will send the wrong keys only if output of a real execution of the protocol and the output of the ideal functionality are different.

Those outputs are identical if all buckets have a majority of fault free gates (this is by construction), and by Lemma 6.2 this happens with overwhelming probability.

□

6.5 Complexity Analysis

We will now sketch a complexity analysis counting the calls to symmetric primitives used in π_{LEGO}. During the protocol we setup $\mathcal{F}^{\sigma,3\Gamma,\ell}_{\text{WCOM}}$, do ℓ Or- and Oblivious-Openings (to deal with inputs), we garbled Γ gates using a call to Yao and each of these are evaluated with a call to Eval. We do a total of $(6\rho+1)s$ regular openings of commitments ($3\rho s$ in **Cut-n-Choose** and $(3\rho + 1)$ in **Soldering**).

Assuming $\mathcal{F}_{\text{WCOM}}$ is implemented using the protocol in Chapter 5 this becomes

$$7(3\Gamma) + 48\sigma\ell + (6\rho + 1)s = 42\rho s + 48\sigma\ell + (6\rho + 1)s$$

calls to enc, and

$$\frac{(4(3\Gamma) + 24\sigma\ell)(n+t)}{\psi} + 14t\log(n) + 14\ell$$
$$= \frac{(24\rho s) + 24\sigma\ell)(n+t)}{\psi} + 14t\log(n) + 14\ell$$

calls to H to handle the commitments. A standard implementation of (Yao, Eval) will use 4 calls to a symmetric cipher to garble each gate and 1 call evaluate it. If we count each of these calls as a call to H, this adds $5\Gamma = 10\rho s$ calls to H.

By Lemma 6.2 we need $\rho \geq \frac{7\sigma+12}{2(\log(s)-7)}$ to get statistical security $2^{-\sigma}$. If for example we let $\sigma = 80$ this gives us $\rho \geq 22$ for circuits of size 2^{20}. Using computational security parameter $\psi = 120$ this means that we get

$$\frac{(24 \cdot 22)(n+t)}{\psi} + 5 \cdot 22 = \frac{(528)(n+t)}{\psi} + 110 \ ,$$

calls to H and

$$42 \cdot 22 = 924$$

calls to enc per gate of the circuit. Based on the discussion in Chapter 5 of the encoding used to implement $\mathcal{F}_{\text{WCOM}}$, this means around 10,000 calls to H.

Setup

1. A and B initialize a $\mathcal{F}_{\text{WCOM}}^{\sigma,3\Gamma+1,\ell}$ functionality.
2. A samples $\Delta \in_R \{0,1\}^\psi$ and inputs $(\texttt{commit}, \mathsf{id}(\Delta), \Delta)$ to $\mathcal{F}_{\text{WCOM}}$.
3. B samples a random $T \subset [\Gamma]$ of size ρs, and for all $i \in T$ samples $u_i, v_i \in_R \{0,1\}$. Let $U = [\Gamma] \setminus T$.
4. B samples $\mathsf{BucketOf} \in \mathcal{B}$ as described in Section 6.3.1.
5. B sends $C_T = \mathsf{Com}(T, (u_i, v_i)_{i\in T}, \mathsf{BucketOf}; r_T)$ to A.
6. For each $j \in \mathsf{Binputs}$, B inputs $(\texttt{OT-choose}, j, b_j)$ to $\mathcal{F}_{\text{WCOM}}$.

Garbling

1. For all $i \in [\Gamma]$, A samples $L_0^i, R_0^i \in_R \{0,1\}^\psi$, computes $(gg_i, O_0^i) \leftarrow \mathsf{Yao}(i, L_0^i, R_0^i, \Delta)$ and sends $GG = (gg_i)_{i\in[\Gamma]}$ to B.
2. A inputs $(\texttt{commit}, \mathsf{id}(L_0^i), L_0^i)$, $(\texttt{commit}, \mathsf{id}(R_0^i), R_0^i)$ and $(\texttt{commit}, \mathsf{id}(O_0^i), O_0^i)$ to $\mathcal{F}_{\text{WCOM}}$.

Cut-n-Choose

1. B sends T, $(u_i, v_i)_{i\in T}$, $\mathsf{BucketOf}$ and randomness r_T to A.
2. If this is not a valid opening of C_T A aborts. Otherwise, for all $i \in T$ A inputs to $\mathcal{F}_{\text{WCOM}}$ $(\texttt{open}, \mathsf{id}(L_{u_i}^i))$, $(\texttt{open}, \mathsf{id}(R_{v_i}^i))$, $(\texttt{open}, \mathsf{id}(O_{u_i\wedge v_i}^i))$. Let $\hat{L}^i, \hat{R}^i, \hat{O}^i$ be the values output to B by $\mathcal{F}_{\text{WCOM}}$.
3. B aborts if there is an $i \in T$ so that $\hat{O}^i \neq \mathsf{Eval}(gg_i, \hat{L}^i, \hat{R}^i)$.

Soldering

1. **Horizontal Soldering:** For all $j \in \mathsf{gates}$, let $h = \mathsf{BucketHead}(j)$: For all $i \neq h \in \mathsf{Bucket}(j)$ A inputs $(\texttt{open}, \{\mathsf{id}(L^h), \mathsf{id}(L^i)\})$, $(\texttt{open}, \{\mathsf{id}(R^h), \mathsf{id}(R^i)\})$, and $(\texttt{open}, \{\mathsf{id}(O^h), \mathsf{id}(O^i)\})$ to $\mathcal{F}_{\text{WCOM}}$. Let dL^i, dR^i, dO^i be the keys output to B from $\mathcal{F}_{\text{WCOM}}$ and $sgg_i = \mathsf{Shift}(gg_i, dL^i, dR^i, dO^i)$.
2. **Vertical Soldering:** For all $j \in \mathsf{gates} \setminus \mathsf{inputGates}$, let $h = \mathsf{BucketHead}(j)$: A inputs $(\texttt{open}, \{\mathsf{id}(L^h)\} \cup \{\mathsf{id}(O^{\mathsf{BucketHead}(i)})\}_{i\in lp(j)})$ and $(\texttt{open}, \{\mathsf{id}(R^h)\} \cup \{\mathsf{id}(O^{\mathsf{BucketHead}(i)})\}_{i\in \mathsf{rp}(j)})$ to $\mathcal{F}_{\text{WCOM}}$. Let dL^h, dR^h be the keys output to B by $\mathcal{F}_{\text{WCOM}}$ and $sgg_h = \mathsf{Shift}(gg_h, dL^h, dR^h, 0^\psi)$.
3. **Input Soldering:** For all $j \in \mathsf{inputGates}$, let $h = \mathsf{BucketHead}(j)$: A inputs $(\texttt{open}, \{\mathsf{id}(L^h), \mathsf{id}(R^h)\})$ to $\mathcal{F}_{\text{WCOM}}$. Let dR^h be the key output to B by $\mathcal{F}_{\text{WCOM}}$ and $sgg_h = \mathsf{Shift}(sgg_h, 0^\psi, dR^h, 0^\psi)$.

Input

1. For all $j \in \mathsf{Ainputs}$ let $h = \mathsf{BucketHead}(j)$, A inputs $(\texttt{OR-open}, \mathsf{id}(L_0^h), \mathsf{id}(L_1^h), a_j)$ to $\mathcal{F}_{\text{WCOM}}$.
2. For all $j \in \mathsf{Binputs}$ let $h = \mathsf{BucketHead}(j)$, A inputs $(\texttt{OT-open}, j, \mathsf{id}(L_0^h), \mathsf{id}(L_1^h))$ to $\mathcal{F}_{\text{WCOM}}$.
3. In both cases let $\hat{L}^h$ be the key output from $\mathcal{F}_{\text{WCOM}}$ to B. B computes a list of candidate keys $\mathsf{Cand}_j = (\mathsf{ShiftEval}(sgg_i, \hat{L}^h, \hat{L}^h))_{i\in\mathsf{Bucket}(j)}$. If any key appears more than $\rho/2$ times in Cand_j name it $\hat{O}^j$, otherwise B aborts.

Evaluate

1. For each gate $j \in \mathsf{gates} \setminus \mathsf{inputGates}$, B computes:
   (a) Left and right keys $\hat{L}^j = \bigoplus_{k\in\mathsf{lp}(j)} \hat{O}^k$ and $\hat{R}^j = \bigoplus_{k\in\mathsf{rp}(j)} \hat{O}^k$.
   (b) The list of output keys $\mathsf{Cand}_j = (\mathsf{ShiftEval}(sgg_i, \hat{L}^j, \hat{R}^j))_{i\in\mathsf{Bucket}(j)}$.
   (c) If a key appears more than $\rho/2$ times in Cand_j name it $\hat{O}^j$ and proceed, otherwise abort.
2. For all $j \in \mathsf{outputGates}$, B sends $\hat{O}^j$ to A.
3. For all $j \in \mathsf{outputGates}$, A outputs $z_j = 0$ if $\hat{O}^j = O_0^{\mathsf{BucketHead}(j)}$, $z_j = 1$ if $\hat{O}^j = O_1^{\mathsf{BucketHead}(j)}$ and aborts otherwise.

Figure 6.3: The Protocol π_{LEGO} Implementing $\mathcal{F}_{\text{SFE}}$.

Bibliography

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In Bernard Chazelle, editor, *ICS*, pages 45–60. Tsinghua University Press, 2011.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security*, pages 257–266. ACM, 2008.

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In Paterson [Pat11], pages 169–188.

[Bea95] Donald Beaver. Precomputing oblivious transfer. In Coppersmith [Cop95], pages 97–109.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In Gary L. Miller, editor, *STOC*, pages 479–488. ACM, 1996.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM Conference on Computer and Communications Security*, pages 784–796. ACM, 2012.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CC06] Hao Chen and Ronald Cramer. Algebraic geometric secret sharing schemes and secure multi-party computations over small fields. In Cynthia Dwork, editor, *CRYPTO*, volume 4117 of *Lecture Notes in Computer Science*, pages 521–536. Springer, 2006.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-xor" technique. In Ronald Cramer, editor, *TCC*, volume 7194 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2012.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In John H. Reif, editor, *STOC*, pages 494–503. ACM, 2002.

[Cop95] Don Coppersmith, editor. *Advances in Cryptology - CRYPTO '95, 15th Annual International Cryptology Conference, Santa Barbara, California, USA, August 27-31, 1995, Proceedings*, volume 963 of *Lecture Notes in Computer Science*. Springer, 1995.

[CvdGT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Coppersmith [Cop95], pages 110–123.

[DKL+12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority mpc protocol. In Ivan Visconti and Roberto De Prisco, editors, *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 241–263. Springer, 2012.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Safavi-Naini and Canetti [SNC12], pages 643–662.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[FJN+13] Tore Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Minilego: Efficient secure two-party computation from general assumptions, 2013.

[FN13] Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and maliciously secure two-party computation using the gpu. Cryptology ePrint Archive, Report 2013/046, 2013. http://eprint.iacr.org/.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[GMY04] Juan A. Garay, Philip MacKenzie, and Ke Yang. Efficient and universally composable committed oblivious transfer and applications. In Moni Naor, editor, *TCC*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 2004.

[Gol09] Oded Goldreich. Comments on eight TCC'09 talks. Manuscript, 2009. http://www.wisdom.weizmann.ac.il/~oded/X/tcc09.ps.

[HEK+11] Yan Huang, David Evans, Jonathan Katz, , and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[HIKN08] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. OT-combiners via secure computation. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 393–411. Springer, 2008.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security and Privacy*, pages 272–284. IEEE Computer Society, 2012.

[HKN+05] Danny Harnik, Joe Kilian, Moni Naor, Omer Reingold, and Alon Rosen. On robust combiners for oblivious transfer and other primitives. In Ronald Cramer, editor, *EUROCRYPT*, volume 3494 of *Lecture Notes in Computer Science*, pages 96–113. Springer, 2005.

[HKS+10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *ACM Conference on Computer and Communications Security*, pages 451–462, 2010.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[IKOS08] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Cryptography with constant computational overhead. In Cynthia Dwork, editor, *STOC*, pages 433–442. ACM, 2008.

[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 572–591. Springer, 2008.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In David S. Johnson, editor, *STOC*, pages 44–61. ACM, 1989.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[KSS12] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21st USENIX conference on Security symposium*, pages 14–14, 2012.

[LOP11] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 259–276. Springer, 2011.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In Moni Naor, editor, *EUROCRYPT*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LP11] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. In Yuval Ishai, editor, *TCC*, volume 6597 of *Lecture Notes in Computer Science*, pages 329–346. Springer, 2011.

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 2–20. Springer, 2008.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302. USENIX, 2004.

[Nie07] Jesper Buus Nielsen. Extending oblivious transfers efficiently - how to get robustness almost for free. Cryptology ePrint Archive, Report 2007/215, 2007. http://eprint.iacr.org/.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Safavi-Naini and Canetti [SNC12], pages 681–700.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. Lego for two-party secure computation. In Reingold [Rei09], pages 368–386.

[Orl11] Claudio Orlandi. *Secure Computation in Untrusted Environments*. PhD thesis, Aarhus Universitet, 2011.

[Pat11] Kenneth G. Paterson, editor. *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*. Springer, 2011.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 250–267. Springer, 2009.

[Rei09] Omer Reingold, editor. *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*. Springer, 2009.

[SNC12] Reihaneh Safavi-Naini and Ran Canetti, editors. *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, volume 7417 of *Lecture Notes in Computer Science*. Springer, 2012.

[SS11] Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Paterson [Pat11], pages 386–405.

[ST04] Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In Pil Joong Lee, editor, *ASIACRYPT*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136. Springer, 2004.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167. IEEE Computer Society, 1986.