A New Approach to Practical Active-Secure Two-Party Computation

Jesper Buus Nielsen[1], Peter Sebastian Nordholt[1], Claudio Orlandi[2], Sai Sheshank Burra[3]

[1] Aarhus University
[2] Bar-Ilan University
[3] Indian Institute of Technology Guwahati

Abstract. We propose a new approach to practical two-party computation secure against an active adversary. All prior practical protocols were based on Yao's garbled circuits. We use an OT-based approach and get efficiency via OT extension in the random oracle model. To get a practical protocol we introduce a number of novel techniques for relating the outputs and inputs of OTs in a larger construction.

We also report on an implementation of this approach, that shows that our protocol is more efficient than any previous one: For big enough circuits, we can evaluate more than 20000 Boolean gates per second. As an example, evaluating one oblivious AES encryption (~ 34000 gates) takes 64 seconds, but when repeating the task 27 times it only takes less than 3 seconds per instance.

Table of Contents

1 Introduction

Secure two-party computation (2PC), introduced by Yao [Yao82], allows two parties to jointly compute any function of their inputs in such a way that 1) the output of the computation is correct and 2) the inputs are kept private. Yao's protocol is secure only if the participants are *semi-honest* (they follow the protocol but try to learn more than they should by looking at their transcript of the protocol). A more realistic security definition considers *malicious adversaries*, that can arbitrarily deviate from the protocol.

A large number of approaches to 2PC have been proposed, falling into three main types, those based on Yao's garbled circuit techniques, those based on some form of homomorphic encryption and those based on oblivious transfer. Recently a number of efforts to implement 2PC in practice have been reported on; In sharp contrast to the theory, almost all of these are based on one type of 2PC, namely Yao's garbled circuit technique. One of the main advantages of Yao's garbled circuits is that it is primarily based on symmetric primitives: It uses one OT per input bit, but then uses only a few calls to, e.g., a hash function per gate in the circuit to be evaluated. The other approaches are heavy on public-key primitives which are typically orders of magnitude slower than symmetric primitives.

However, in 2003 Ishai *et al.* introduced the idea of extending OTs *efficiently* [IKNP03]—their protocol allows to turn κ seed OTs based on public-key crypto into any polynomial $\ell = \text{poly}(\kappa)$ number of OTs using only $O(\ell)$ invocations of a cryptographic hash function. For big enough ℓ the cost of the κ seed OTs is amortized away and OT extension essentially turns OT into a symmetric primitive in terms of its computational complexity. Since the basic approach of basing 2PC on OT in [GMW87] is efficient in terms of consumption of OTs and communication, this gives the hope that OT-based 2PC too could be practical. This paper reports on the first implementation made to investigate the practicality of OT-based 2PC.

Our starting point is the efficient passive-secure OT extension protocol of [IKNP03] and passive-secure 2PC of [GMW87]. In order to get active security and preserve the high practical efficiency of these protocols we chose to develop substantially different techniques, differentiating from other works that were only interested in *asymptotic* efficiency [HIKN08,Nie07,IPS08]. We report a number of contributions to the theory and practice of 2PC:

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 protocol in [HIKN08], but it is only a small factor slower than the passive-secure protocol in [IKNP03].
2. We give the first implementation of the idea of extending OTs efficiently. The protocol is active-secure and generates 500,000 OTs per second, showing that implementations needing a large number of OTs can be practical.
3. 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.
4. We give the first implementation of practical 2PC not based on Yao's garbled circuit technique. Introducing a new practical technique is a significant contribution to the field in itself. In addition, our protocol shows favorable timings compared to the Yao-based implementations.

1.1 Comparison with Related Work

The question on the *asymptotical* computational overhead of cryptography was (essentially) settled in [IKOS08]. On the other hand, there is growing interest in understanding the *practical* overhead of secure computation, and several works have perfected and implemented protocols based on Yao garbled circuits [MNPS04,BDNP08,LPS08,KS08,PSSW09,HKS+10,MK10,LP11,SS11,HEK+11], protocols based on homomorphic encryption [IPS09,DO10,JMN10,BDOZ11] and protocols based on OT [IPS08,LOP11,CHK+11].

		Security	Model	Rounds	Time
(a)	DK [DK10] (3 parties)	Passive	SM	$O(d)$	1.5s
(b)	DK [DK10] (4 parties)	Active	SM	$O(d)$	4.5s
(c)	sS [SS11]	Active	SM	$O(1)$	192s
(d)	HEKM [HEK+11]	Passive	ROM	$O(1)$	0.2s
(e)	IPS-LOP [IPS08,LOP11]	Active	SM	$O(d)$	79s
(f)	This (single)	Active	ROM	$O(d)$	64s
(g)	This (27, amortized)	Active	ROM	$O(d)$	2.5s

Table 1. Brief comparison with other implementations.

A brief comparison of the time needed for oblivious AES evaluation for the best known implementations are shown in Table 1.[4] The protocols in rows (a-b) are for 3 and 4 parties respectively, and are secure against at most one corrupted party. One of the goals of the work in row (c) is how to efficiently support different outputs for different parties: in our OT based protocol this feature comes for free. The time in row (e) is an estimate made by [LOP11] on the running time of their optimized version of the OT-based protocol in [IPS08]. 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 proven secure in the standard model (SM) or the random oracle model (ROM).

The significance of this work is shown in row (g). The reason for the dramatic drop between row (f) and (g) is that in (f), when we only encrypt one block, our implementation preprocesses for many more gates than is needed, for ease of implementation. In (g) we encrypt 27 blocks, which is the minimum value which eats to up all the preprocessed values. We consider these results positive: our implementation is as fast or faster than any other 2PC protocol, even when encrypting only one block. And more importantly, when running at full capacity, the price to pay for active security is about a factor 10 against the passive-secure protocol in (d). We stress that this is only a limited comparison, as the different experiments were run on different hardware and network setups: when several options were available, we selected the best time reported by the other implementations. See Sect. 7 for more timings and details of our implementation.

1.2 Overview of Our Approach

We start from a classic textbook protocol for two-party computation [Gol04, Sec. 7.3]. In this protocol, Alice holds secret shares x_A, y_A and Bob holds secret shares x_B, y_B of some bits x, y s.t. $x_A \oplus x_B = x$ and $y_A \oplus y_B = y$. Alice and Bob want to compute secret shares of $z = g(x, y)$ where g is some Boolean gate, for instance the AND gate: Alice and Bob 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 oblivious transfer (OT) to compute the cross products ($x_A y_B$ and $x_B y_A$). Now the parties can iterate for the next layer of the circuit, up to the end where they will reconstruct the output values by revealing their shares.

[4] 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].

This protocol is secure against a semi-honest adversary: assuming the OT protocol to be secure, Alice and Bob 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* $\Delta \in \{0,1\}^\kappa$. 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* ($K_x, K_y \in \{0,1\}^\kappa$) 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 (see Fig. 1 for the main structure):

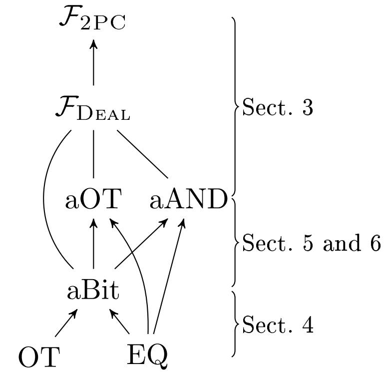

Fig. 1. Paper outline. This order of presentation is chosen to allow the best progression in introduction of our new techniques.

Bit Authentication: We first implement oblivious authentication of bits (aBit). As detailed in Sect. 4, to construct authenticated bits we start by extending a few (say $\kappa = 640$) seed $\binom{2}{1}$-OTs into many (say $\ell = 2^{20}$) OTs, using OT extension. Then, if A wants to get a bit x authenticated, she can input it as the choice bit in an OT, while B can input $(K_x, K_x \oplus \Delta)$, playing the sender in the OT. Now A receives $M_x = K_x \oplus x\Delta$. It should, of course, be ensured that even a corrupted B uses the same value Δ in all OTs. I.e., it should hold for all produced OTs that the XORs of the offered message pairs are constant—this constant value is then taken to be Δ. It turns out, however, that when using the highly efficient *passive-secure* 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 we note that for this use the protocol in [IKNP03] happens to be *active-secure*! Using cut-and-choose 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. This cut-and-choose technique uses one call to a box EQ for checking equality.

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, again using the box EQ once. 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 aBits representing the sender messages x_0, x_1 and the receiver choice bit c. To produce the receiver's output, first a random aBit is sampled. Then this bit is "corrected" in order to be consistent with the run of an OT protocol with input messages

x_0, 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. One call to the box EQ is used.

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,Gar04]). 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 result 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, we get active security paying only a constant overhead over the passive-secure protocol in [IKNP03]. In the generation of aANDs and aOTs, we replace cut-and-choose with efficient, slightly leaky proofs and then use a combiner to get rid of the leakage: When we preprocess for ℓ gates and combine B leaky objects to get each potentially unleaky object, the probability of leaking is $(2\ell)^{-B} = 2^{-\log_2(\ell)(B-1)}$. As an example, if we preprocess for 2^{20} gates with an overhead of $B = 6$, then we get leakage probability 2^{-100}.

As a corollary to being able to generate any $\ell = \text{poly}(\kappa)$ active-secure aBits from $O(\kappa)$ seed OTs and $O(\ell)$ calls to a hash-function, we get that we can generate any $\ell = \text{poly}(\kappa)$ active-secure $\binom{2}{1}$-OTs of κ-bit strings from $O(\kappa)$ seed OTs and $O(\ell)$ calls to a hash-function, matching the asymptotic complexity of [HIKN08] while dramatically reducing their hidden constants.

2 Preliminaries and Notation

We use κ (and sometimes ψ) to denote the security parameter. We require that a poly-time adversary break the protocol with probability at most $\text{poly}(\kappa)2^{-\kappa}$. For a bit-string $S \in \{0,1\}^*$ we define $0S \stackrel{\text{def}}{=} 0^{|S|}$ and $1S \stackrel{\text{def}}{=} S$. 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.

The UC Framework We prove our results static, active-secure in the UC framework [Can01], and we assume the reader to be familiar with it. We will idiosyncratically use the word *box* instead of the usual term *ideal functionality*. To simplify the statements of our results we use the following terminology:

Definition 1. *We say that a box* A *is* reducible *to a box* B *if there exist an actively secure implementation* π *of* A *which uses only one call to* B*. We say that* A *is* locally *reducible to* B *if the parties of* π *do not communicate (except through the one call to* B*). We say that* A *is* linear *reducible to* 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 A is (linear, locally) reducible to B and B is (linear, locally) reducible to C, then A is (linear, locally) reducible to C.

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

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 assumption is an interesting theoretical question, see [AHI10,CKKZ11].

Oblivious Transfer We use a box $\mathrm{OT}(\tau, \ell)$ which can be used to perform τ $\binom{2}{1}$-oblivious transfers of strings of bit-length ℓ. In each of the τ OTs the sender S has two inputs $x_0, x_1 \in \{0,1\}^\ell$, called the *messages*, and the receiver R has an input $c \in \{0,1\}$, called the *choice bit*. The output to R is $x_c = c(x_0 \oplus x_1) \oplus x_0$. No party learns any other information.

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

For completeness we give a protocol which securely implements EQ in the RO model. Let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a hash function, modeled as a RO. Let κ be the security parameter.

1. A chooses a random string $r \in_R \{0,1\}^\kappa$, computes $c = H(x||r)$ and sends it 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 $\mathrm{EQ}(\ell)$ 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 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 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 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 $\mathrm{poly}(\kappa)2^{-\kappa}$, 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 EQ box and receives (x, y). Now, it chooses a random $r \in_R \{0,1\}^\kappa$ 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 $\mathrm{poly}(\kappa)2^{-\kappa}$, as $r \in \{0,1\}^\kappa$ is uniformly random, and the environment is only allowed a polynomial number of calls to the RO.

Leakage Functions We use a notion of a class $\mathcal{L}$ of *leakage functions on τ bits.* The context is that there is some uniformly random secret value $\Delta \in_{\mathrm{R}} \{0,1\}^\tau$ and some adversary A wants to guess Δ. To aid A, she can do an attack which might leak some of the bits of Δ. The attack, however, might be detected. Each $L \in \mathcal{L}$ is a poly-time sampleable distribution on $(S,c) \in 2^{\{1,\dots,\tau\}} \times \{0,1\}$. Here c specifies if the attack was detected, where $c=0$ signals detection, and S specifies the bits to be leaked if the attack was not detected. We need a measure of how many bits a class $\mathcal{L}$ leaks. We do this via a game for an unbounded adversary A.

1. The game picks a uniformly random $\Delta \in_{\mathrm{R}} \{0,1\}^\tau$.
2. A inputs $L \in \mathcal{L}$.
3. The game samples $(S,c) \leftarrow L$. If $c=0$, A loses. If $c=1$, the game gives $\{(i,\Delta_i)\}_{i\in S}$ to A.
4. Let $\overline{S} = \{1,\dots,\tau\}\backslash S$. A inputs the guesses $\{(i,g_i)\}_{i\in\overline{S}}$. If $g_i = \Delta_i$ for all $i \in \overline{S}$, A wins, otherwise she loses.

We say that an adversary A is *optimal* if she has the highest possible probability of winning the game above. If there were no leakage, i.e., $S=\emptyset$, then it is clear that the 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 the optimal A can win the game with probability exactly $2^{s-\tau}$. This motivates defining the number of bits leaked by $\mathcal{L}$ to be $\mathrm{leak}_{\mathcal{L}} \stackrel{\mathrm{def}}{=} \log_2(\mathrm{success}_{\mathcal{L}}) + \tau$, where $\mathrm{success}_{\mathcal{L}}$ is the probability that the optimal A wins the game. It is easy (details below) to see that if we take expectation over random (S,c) sampled from L, then $\mathrm{leak}_{\mathcal{L}} = \max_{L\in\mathcal{L}} \log_2\left(\mathrm{E}\left[c2^{|S|}\right]\right)$.

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

We now rewrite the definition of $\mathrm{leak}_{\mathcal{L}}$ to make it more workable.

It is clear that the optimal A can guess all Δ_i for $i \in \overline{S}$ with probability exactly $2^{|S|-\tau}$. This means that the optimal A wins with probability $\sum_{s=0}^{\tau} \Pr\left[(S,c) \leftarrow L : |S| = s \wedge c = 1\right] 2^{s-\tau}$. 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 $|S|=s$. Note that $I_s = cJ_s$ and that $\sum_s J_s 2^s = 2^{|S|}$. So, if we take expectation over (S,c) sampled from L, then we get that

$$
\begin{aligned}
\sum_{s=0}^{\tau} \Pr\left[(S,c) \leftarrow L : |S| = s \wedge c = 1\right] 2^s &= \sum_{s=0}^{\tau} \mathrm{E}\left[I_s\right] 2^s \\
&= \mathrm{E}\left[\sum_{s=0}^{\tau} I_s 2^s\right] = \mathrm{E}\left[\sum_{s=0}^{\tau} cJ_s 2^s\right] \\
&= \mathrm{E}\left[c\sum_{s=0}^{\tau} J_s 2^s\right] = \mathrm{E}\left[c2^{|S|}\right] \ .
\end{aligned}
$$

Hence $\mathrm{success}_L = 2^{-\tau}\,\mathrm{E}\left[c2^{|S|}\right]$ is the probability of winning when using L and playing optimal. Hence $\mathrm{success}_{\mathcal{L}} = \max_{L\in\mathcal{L}}(2^{-\tau}\,\mathrm{E}\left[c2^{|S|}\right])$ and $\log_2(\mathrm{success}_{\mathcal{L}}) = -\tau + \log_2 \max_{L\in\mathcal{L}}\left(\mathrm{E}\left[c2^{|S|}\right]\right)$, which shows that

$$
\mathrm{leak}_{\mathcal{L}} = \max_{L\in\mathcal{L}} \log_2\left(\mathrm{E}\left[c2^{|S|}\right]\right) \ ,
$$

as claimed above.

3 The Two-Party Computation Protocol

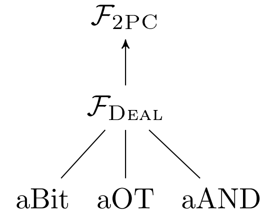

Fig. 2. Sect. 3 outline.

We want to implement the box $\mathcal{F}_{\mathrm{2PC}}$ for Boolean two-party secure computation as described in Fig. 4. We will implement this box in the $\mathcal{F}_{\mathrm{DEAL}}$-hybrid model of Fig. 5. This box provides the parties with aBits, aANDs and aOTs, and models the preprocessing phase of our protocol. We introduce notation in Fig. 3 for working with authenticated bits. The protocol implementing $\mathcal{F}_{\mathrm{2PC}}$ in the dealer model is described in Fig. 6. 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 box) are linearly reducible to the randomized version, as can be easily deduced from the protocol description. The following result is proven in App. B:

Theorem 1. *The protocol in Fig. 6 securely implements the box $\mathcal{F}_{\mathrm{2PC}}$ in the $\mathcal{F}_{\mathrm{DEAL}}$-hybrid model with security parameter κ.*

Global Key: We call $\Delta_A, \Delta_B \in \{0,1\}^\kappa$ 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 $K_x \in \{0,1\}^\kappa$ and A knows a bit x and a MAC $M_x = K_x \oplus x\Delta_A \in \{0,1\}^\kappa$. 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 $[y]_{\mathsf{A}}$.

It is possible to authenticate a constant bit (a value known both to A and B) $b \in \{0,1\}$ as follows: A sets $M_b = 0^\kappa$, 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 $[y]$.

It is possible to create an authenticated share of a constant $b \in \{0,1\}$ as follows: A and B create $[b] = [b|0]$. For a constant value $b \in \{0,1\}$, 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 $M_x = K_x \oplus \Delta_A$.

Fig. 3. Notation for authenticated and shared bits.

Why the global key queries? The $\mathcal{F}_{\mathrm{DEAL}}$ box (Fig. 5) 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 $\mathcal{F}_{\mathrm{DEAL}}$ is proven UC secure, the environment has access to either $\mathcal{F}_{\mathrm{DEAL}}$ or the protocol implementing $\mathcal{F}_{\mathrm{DEAL}}$. In both cases the environment learns the global keys Δ_A and Δ_B. In particular,

Rand: On input (rand, vid) from A and B, with vid a fresh identifier, the box picks $r \in_R \{0,1\}$ and stores (vid, r).
Input: On input (input, P, vid, x) from $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ and (input, P, vid, ?) from the other party, with vid a fresh identifier, the box stores (vid, x).
XOR: On command (xor, vid_1, vid_2, vid_3) from both parties (if vid_1, vid_2 are defined and vid_3 is fresh), the box retrieves (vid_1, x), (vid_2, y) and stores $(vid_3, x \oplus y)$.
AND: As **XOR**, but store $(vid_3, x \cdot y)$.
Output: On input (output, P, vid) from both parties, with $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}\}$ (and vid defined), the box retrieves (vid, x) and outputs it to P.

At each command the box 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.

Fig. 4. The box $\mathcal{F}_{\text{2PC}}$ for Boolean Two-party Computation.

Initialize: On input (init) from A and (init) from B, the box samples $\Delta_A, \Delta_B \in \{0,1\}^\kappa$, 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 box 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 box sets $K_x = M_x \oplus x\Delta_A$.
Authenticated Bit (B): On input (aBIT, B) from A and B, the box 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 box 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 box 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.

Fig. 5. The box $\mathcal{F}_{\text{DEAL}}$ for dealing preprocessed values.

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 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 $\mathcal{F}_{\text{DEAL}}$ (Fig. 6), 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 Fig. 6, 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 $N = 0^\kappa$ and the key holder, e.g. B, sets $N' = 0^\kappa$. 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

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.

Fig. 6. Protocol for $\mathcal{F}_{\text{2PC}}$ in the $\mathcal{F}_{\text{DEAL}}$-hybrid model

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 $O(\kappa|C|)$ to $O(|C| + o\kappa)$, where o is the number of rounds in which outputs are opened. For a circuit of depth $O(|C|/\kappa)$, the communication is $O(|C|)$.

Implementing $\mathcal{F}_{\text{DEAL}}$. In the following sections we show how to implement $\mathcal{F}_{\text{DEAL}}$. In Sect. 4 we implement just the part with the commands **Authenticated Bits**. In Sect. 5 we show how to extend with the **Authenticated OT** commands, by showing how to implement many aOTs from many aBits. In Sect. 6 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 both 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 $\mathcal{F}_{\text{DEAL}}$.

4 Bit Authentication

In this section we show how to efficiently implement (oblivious) bit authentication, i.e., we want to be in a situation where A knows some bits $x_1, \ldots, x_\ell$ together with MACs $M_1, \ldots, M_\ell$, while B holds a global key Δ_A and local keys $K_1, \ldots, K_\ell$ s.t. $M_i = K_i \oplus x_i \Delta_A$, as described in $\mathcal{F}_{\text{DEAL}}$ (Fig. 5). Given the complete symmetry of $\mathcal{F}_{\text{DEAL}}$, we only describe the case where A is MAC holder.

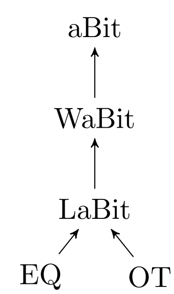

Fig. 7. Sect. 4 outline.

If the parties were honest, we could do the following: A and B run an OT where B inputs the two messages $(K_i, K_i \oplus \Delta_A)$ and A chooses x_i, to receive $M_i = K_i \oplus x_i \Delta_A$. However, if B is dishonest he might not use the same Δ_A in all OTs. The main ideas that make the protocol secure against cheating parties are the following:

1. For reasons that will be apparent later, we will actually start in the opposite direction and let B receive some authenticated bits y_i using an OT, where A is supposed to always use the same

global key Γ_B. Thus an honest A inputs $(L_i, L_i \oplus \Gamma_B)$ in the OTs and B receives $N_i = L_i \oplus y_i \Gamma_B$. To check that A is playing honest in most OTs, the authenticated bits are randomly paired and a check is performed, which restricts A to cheat in at most a few OTs.
2. We then notice that what A gains by using different Γ_B's in a few OTs is no more than learning a few of B's bits y_i. We call this a leaky aBit, or LaBit.
3. We show how to turn this situation into an equivalent one where A (not B) receives authenticated random bits x_i's (none of which leaks to B) under a "slightly insecure" global key Γ_A. The insecurity comes from the fact that the leakage of the y_i's turns into the leakage of a few bits of the global key Γ_A towards A. We call this an aBit with weak global key, or WaBit.
4. Using privacy amplification, we amplify the previous setting to a new one where A receives authenticated bits under a (shorter) fully secure global key Δ_A, where no bits of Δ_A are known to A, finally implementing the aBit command of the dealer box.

We will proceed in reverse order and start with step 4 in the previous description: we will start with showing how we can turn authenticated bits under an "insecure" global key Γ_A into authenticated bits under a "secure" (but shorter) global key Δ_A.

4.1 Bit Authentication with Weak Global Key (WaBit)

We will first define the box providing bit authentication, but where some of the bits of the global key might leak. We call this box WaBit (bit authentication with weak global key) and we formally describe it in Fig. 8. The box $\text{WaBit}^{\mathcal{L}}(\ell, \tau)$ outputs ℓ bits with keys of length τ. The box is also parametrized by a class $\mathcal{L}$ of leakage functions on τ bits. The box $\text{aBit}(\ell, \psi)$ is the box $\text{WaBit}^{\mathcal{L}}(\ell, \psi)$ where $\mathcal{L}$ is the class of leakage functions that never leak.

Honest Parties:
1. The box samples $\Gamma_A \in_{\text{R}} \{0,1\}^\tau$ and outputs it to B.
2. The box samples and outputs $[x_1]_{\mathsf{A}}, \ldots, [x_\ell]_{\mathsf{A}}$. Each $[x_i]_{\mathsf{A}} = (x_i, M'_i, K'_i) \in \{0,1\}^{1+2\tau}$ s.t. $M'_i = K'_i \oplus x_i \Gamma_A$.

Corrupted Parties:
1. If A is corrupted, then A may choose a leakage function $L \in \mathcal{L}$. Then the box samples $(S, c) \leftarrow L$. If $c = 0$ the box outputs `fail` to B and terminates. If $c = 1$, the box outputs $\{(i, (\Gamma_A)_i)\}_{i \in S}$ to A.
2. If A is corrupted, then A chooses the x_i and the M'_i and then $K'_i = M'_i \oplus x_i \Gamma_A$.
3. If B is corrupted, then B chooses Γ_A and the K'_i.

Global Key Queries: The adversary can input Γ and will be told if $\Gamma = \Gamma_A$.

Fig. 8. The box $\text{WaBit}^{\mathcal{L}}(\ell, \tau)$ for Bit Authentication with Weak Global Key

1. The parties invoke $\text{WaBit}^{\mathcal{L}}(\ell, \tau)$ with $\tau = \frac{22}{3}\psi$. The output to A is $((M'_1, x_1), \ldots, (M'_\ell, x_\ell))$. The output to B is $(\Gamma_A, K'_1, \ldots, K'_\ell)$.
2. B samples $\mathbf{A} \in_{\text{R}} \{0,1\}^{\psi \times \tau}$, a random binary matrix with ψ rows and τ columns, and sends $\mathbf{A}$ to A.
3. A computes $M_i = \mathbf{A} M'_i \in \{0,1\}^\psi$ and outputs $((M_1, x_1), \ldots, (M_\ell, x_\ell))$.
4. B computes $\Delta_A = \mathbf{A}\Gamma_A$ and $K_i = \mathbf{A} K'_i$ and outputs $(\Delta_A, K_1, \ldots, K_\ell)$.

Fig. 9. Subprotocol for reducing $\text{aBit}(\ell, \psi)$ to $\text{WaBit}^{\mathcal{L}}(\ell, \tau)$.

In Fig. 9 we describe a protocol which takes a box WaBit, where one quarter of the bits of the global key might leak, and amplifies it to a box aBit where the global key is perfectly secret. The protocol is described for general $\mathcal{L}$ and it is parametrized by a desired security level ψ. The proof of the following theorem can be found in App. C.

Theorem 2. *Let $\tau = \frac{22}{3}\psi$ and $\mathcal{L}$ be a $\left(\frac{3}{4}\tau\right)$-secure leakage function on τ bits. The protocol in Fig. 9 securely implements* $\mathrm{aBit}(\ell, \psi)$ *in the* $\mathrm{WaBit}^{\mathcal{L}}(\ell, \tau)$*-hybrid model with security parameter ψ. The communication is $O(\psi^2)$ and the work is $O(\psi^2 \ell)$.*

4.2 Bit Authentication with Leaking Bits (LaBit)

We now show another insecure box for aBit. The new box is insecure in the sense that a few of the bits to be authenticated might leak to the other party. We call this box an aBit with leaking bits, or LaBit and formally describe it in Fig. 10. The box $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$ outputs τ authenticated bits with keys of length ℓ, and is parametrized by a class of leakage functions $\mathcal{L}$ on τ-bits. We show that $\mathrm{WaBit}^{\mathcal{L}}$ can be reduced to $\mathrm{LaBit}^{\mathcal{L}}$. In the reduction, a LaBit that outputs authenticated bits $[y_i]_{\mathsf{B}}$ to B can be turned into a WaBit that outputs authenticated bits $[x_j]_{\mathsf{A}}$ to A, therefore we present the LaBit box that outputs bits to B. The reduction is strongly inspired by the OT extension techniques in [IKNP03].

Honest Parties:

1. The box samples $\Gamma_B \in_{\mathrm{R}} \{0,1\}^{\ell}$ and outputs it to A.
2. The box samples and outputs $[y_1]_{\mathsf{B}}, \ldots, [y_\tau]_{\mathsf{B}}$. Each $[y_i]_{\mathsf{B}} = (y_i, N_i, L_i) \in \{0,1\}^{1+2\ell}$ s.t. $N_i = L_i \oplus y_i \Gamma_B$.

Corrupted Parties:

1. If A is corrupted, then A may input a leakage function $L \in \mathcal{L}$. Then the box samples $(S, c) \leftarrow L$. If $c = 0$ the box outputs `fail` to B and terminates. If $c = 1$, the box outputs $\{(i, y_i)\}_{i \in S}$ to A.
2. Corrupted parties get to specify their outputs as in Fig. 8.

Choice Bit Queries: The adversary can input Δ and will be told if $\Delta = (y_1, \ldots, y_\tau)$.

Fig. 10. The box $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$ for Bit Authentication with Leaking Bits

1. A and B invoke $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$. B learns $((N_1, y_1), \ldots, (N_\tau, y_\tau))$ and A learns $(\Gamma_B, L_1, \ldots, L_\tau)$.
2. A lets x_j be the j-th bit of Γ_B and M_j the string consisting of the j-th bits from all the strings L_i, i.e. $M_j = L_{1,j} || L_{2,j} || \ldots || L_{\ell,j}$.
3. B lets Γ_A be the string consisting of all the bits y_i, i.e. $\Gamma_A = y_1 || y_2 || \ldots || y_\ell$, and lets K_j be the string consisting of the j-th bits from all the strings N_i, i.e. $K_j = N_{1,j} || N_{2,j} || \ldots || N_{\ell,j}$.
4. A and B now hold $[x_j]_{\mathsf{A}} = (x_j, M_j, K_j)$ for $j = 1, \ldots, \ell$.

Fig. 11. Subprotocol for reducing $\mathrm{WaBit}^{\mathcal{L}}(\ell, \tau)$ to $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$

Theorem 3. *For all ℓ, τ and $\mathcal{L}$ the boxes* $\mathrm{WaBit}^{\mathcal{L}}(\ell, \tau)$ *and* $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$ *are linear locally equivalent, i.e., can be implemented given the other in linear time without interaction.*

Proof. The first direction (reducing WaBit to LaBit) is shown in Fig. 11. The other direction (LaBit is linear locally reducible to WaBit) will follow by the fact that the local transformations are reversible in linear time. One can check that for all $j = 1, \ldots, \tau$, $[x_j]_{\mathsf{A}}$ is a correct authenticated bit. Namely, from the box LaBit we get that for all $i = 1, \ldots, \ell$, $N_i = L_i \oplus y_i \Gamma_B$. In particular the j-th bit satisfies $N_{i,j} = L_{i,j} \oplus y_i (\Gamma_B)_j$, which can be rewritten (using the same renaming as in the protocol) as $K_{j,i} = M_{j,i} \oplus (\Gamma_A)_i x_j$, and therefore $M_j = K_j \oplus x_j \Gamma_A$, as we want. It is easy so see (as the protocol only consists of renamings) that leakage on the choice bits is equivalent to leakage on the global key under this transformation, and guesses on Γ_A are equivalent to guesses on $(y_1, \ldots, y_\tau)$, so giving a simulation argument is straight-forward when $\mathcal{L}$ is the same for both boxes. $\square$

Note that since we turn $\text{LaBit}^{\mathcal{L}}(\ell,\tau)$ into $\text{WaBit}^{\mathcal{L}}(\tau,\ell)$, if we choose $\ell = \text{poly}(\psi)$ we can turn a relatively small number ($\tau = \frac{22}{3}\psi$) of authenticated bits towards one player into a very larger number (ℓ) of authenticated bits towards the other player.

4.3 A Protocol For Bit Authentication With Leaking Bits

In this section we show how to construct authenticated bits starting from OTs. The protocol ensures that most of the authenticated bits will be kept secret, as specified by the LaBit box in Fig. 10.

The main idea of the protocol, described in Fig. 12, is the following: many authenticated bits $[y_i]_{\mathsf{B}}$ for B are created using OTs, where A is supposed to input messages $(L_i, L_i \oplus \Gamma_B)$. To check that A is using the same Γ_B in every OT, the authenticated bits are randomly paired. Given a pair of authenticated bits $[y_i]_{\mathsf{B}}, [y_j]_{\mathsf{B}}$, A and B compute $[z_i]_{\mathsf{B}} = [y_i]_{\mathsf{B}} \oplus [y_j]_{\mathsf{B}} \oplus d_i$ where $d_i = y_i \oplus y_j$ is announced by B. If A behaved honestly, she knows the MAC that B holds on z_i, otherwise she has 1 bit of entropy on this MAC, as shown below. The parties can check if A knows the MAC using the EQ box described in App. 2. As B reveals $y_i \oplus y_j$, they waste $[y_j]_{\mathsf{B}}$ and only use $[y_i]_{\mathsf{B}}$ as output from the protocol—as y_j is uniformly random $y_i \oplus y_j$ leaks no information on y_i. Note that we cannot simply let A reveal the MAC on z_i, as a malicious B could announce $1 \oplus z_i$: this would allow B to learn a MAC on z_i and $1 \oplus z_i$ at the same time, thus leaking Γ_B. Using EQ forces a thus cheating B to guess the MAC on a bit which he did not see, which he can do only with negligible probability $2^{-\ell}$.

1. A samples $\Gamma_B \in_{\mathrm{R}} \{0,1\}^\ell$ and for $i = 1, \dots, \mathcal{T}$ samples $L_i \in_{\mathrm{R}} \{0,1\}^\ell$, where $\mathcal{T} = 2\tau$.
2. B samples $(y_1, \dots, y_{\mathcal{T}}) \in_{\mathrm{R}} \{0,1\}^{\mathcal{T}}$.
3. They run $\mathcal{T}$ OTs, where for $i = 1, \dots, \mathcal{T}$ party A offers $(Y_{i,0}, Y_{i,1}) = (L_i, L_i \oplus \Gamma_B)$ and B selects y_i and receives $N_i = Y_{i,y_i} = L_i \oplus y_i \Gamma_B$. Let $[y_1]_{\mathsf{B}}, \dots, [y_{\mathcal{T}}]_{\mathsf{B}}$ be the candidate authenticated bits produced so far.
4. B picks a uniformly random pairing π (a permutation $\pi : \{1, \dots, \mathcal{T}\} \to \{1, \dots, \mathcal{T}\}$ where $\forall i, \pi(\pi(i)) = i$), and sends π to A. Given a pairing π, let $\mathcal{S}(\pi) = \{i | i \le \pi(i)\}$, i.e., for each pair, add the smallest index to $\mathcal{S}(\pi)$.
5. For all τ indices $i \in \mathcal{S}(\pi)$:
   (a) B announces $d_i = y_i \oplus y_{\pi(i)}$.
   (b) A and B compute $[z_i]_{\mathsf{B}} = [y_i]_{\mathsf{B}} \oplus [y_{\pi(i)}]_{\mathsf{B}} \oplus d_i$.
   (c) Let Z_i and W_i be the MAC and the local key for z_i held by A respectively B. They compare these using EQ and abort if they are different.

   The τ comparisons are done using *one* call on the $\tau\ell$-bit strings $(Z_i)_{i \in \mathcal{S}(\pi)}$ and $(W_i)_{i \in \mathcal{S}(\pi)}$.
6. For all $i \in \mathcal{S}(\pi)$ A and B output $[y_i]_{\mathsf{B}}$.

Fig. 12. The protocol for reducing $\text{LaBit}(\tau,\ell)$ to $\text{OT}(2\tau,\ell)$ and $\text{EQ}(\tau\ell)$.

Note that if A uses different Γ_B in two paired instances, Γ_i and Γ_j say, then the MAC held by B on $y_i \oplus y_j$ (and therefore also z_i) is $(L_i \oplus y_i \Gamma_i) \oplus (L_j \oplus y_j \Gamma_j) = (L_i \oplus L_j) \oplus (y_i \oplus y_j)\Gamma_j \oplus y_i(\Gamma_i \oplus \Gamma_j)$. Since $(\Gamma_i \oplus \Gamma_j) \neq 0^\ell$ and $y_i \oplus y_j$ is fixed by announcing d_i, guessing this MAC is equivalent to guessing y_i. As A only knows $L_i, L_j, \Gamma_i, \Gamma_j$ and $y_i \oplus y_j$, she cannot guess y_i with probability better than $1/2$. Therefore, if A cheats in many OTs, she will get caught with high probability. If she only cheats on a few instances she might pass the test. Doing so confirms her guess on y_i in the pairs where she cheated. Now assume that she cheated in instance i and offered $(L_i, L_i \oplus \Gamma'_B)$ instead of $(L_i, L_i \oplus \Gamma_B)$. After getting her guess on y_i confirmed she can explain the run as an honest run: If $y_i = 0$, the run is equivalent to having offered $(L_i, L_i \oplus \Gamma_B)$, as B gets no information on the second message when $y_i = 0$. If $y_i = 1$, then the run is equivalent to having offered $(L'_i, L'_i \oplus \Gamma_B)$ with

$L_i' = L_i \oplus (\Gamma_B \oplus \Gamma_B')$, as $L_i' \oplus \Gamma_B = L_i \oplus \Gamma_B$ and B gets no information on the first message when $y_i = 1$. So, any cheating strategy of A can be simulated by letting her honestly use the same Γ_B in all pairs and then let her try to guess some bits y_i. If she guesses wrong, the deviation is reported to B. If she guesses right, she is told so and the deviation is not reported to B. This, in turn, can be captured using some appropriate class of leakage functions $\mathcal{L}$. Nailing down the exact $\mathcal{L}$ needed to simulate a given behavior of A, including defining what is the "right" Γ_B, and showing that the needed $\mathcal{L}$ is always κ-secure is a relatively straight-forward but very tedious business. The proof of the following theorem can be found in App. D.

Theorem 4. *Let $\kappa = \frac{3}{4}\tau$, and let $\mathcal{L}$ be a κ secure leakage function on τ bits. The protocol in Fig. 12 securely implements $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$ in the $(\mathrm{OT}(2\tau, \ell), \mathrm{EQ}(\tau\ell))$-hybrid model. The communication is $O(\tau^2)$. The work is $O(\tau\ell)$.*

Corollary 1. *Let ψ denote the security parameter and let $\ell = \mathrm{poly}(\psi)$. The box $\mathrm{aBit}(\ell, \psi)$ can be reduced to $(\mathrm{OT}(\frac{44}{3}\psi, \psi), \mathrm{EQ}(\psi))$. The communication is $O(\psi\ell + \psi^2)$ and the work is $O(\psi^2\ell)$.*

Proof. Combining the above theorems we have that $\mathrm{aBit}(\ell, \psi)$ can be reduced to $(\mathrm{OT}(\frac{44}{3}\psi, \ell), \mathrm{EQ}(\frac{22}{3}\psi\ell))$ with communication $O(\psi^2)$ and work $O(\psi^2\ell)$. For any polynomial ℓ, we can implement $\mathrm{OT}(\frac{44}{3}\psi, \ell)$ given $\mathrm{OT}(\frac{44}{3}\psi, \psi)$ and a pseudo-random generator $\mathrm{prg} : \{0,1\}^\psi \to \{0,1\}^\ell$. 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 κ bits as $O(\kappa)$ work, then the work is $O(\ell\psi)$. We can implement $\mathrm{EQ}(\frac{22}{3}\psi\ell)$ by comparing short hashes produced using the RO. The work is $O(\psi\ell)$. □

Since the oracles $(\mathrm{OT}(\frac{44}{3}\psi, \psi), \mathrm{EQ}(\psi))$ are independent of ℓ, the cost of essentially any reasonable implementation of them can be amortized away by picking ℓ large enough. See App. A for a more detailed complexity analysis.

Efficient OT Extension: We notice that the WaBit box resembles an intermediate step of the OT extension protocol of [IKNP03]. Completing their protocol (i.e., "hashing away" the fact that all messages pairs have the same XOR), gives an efficient protocol for OT extension, with the same asymptotic complexity as [HIKN08], but with dramatically smaller constants. See App. E for details.

5 Authenticated Oblivious Transfer

In this section we show how to implement aOTs. We implemented aBits in Sect. 4, so what remains is to show how to implement aOTs from aBits i.e., to implement the $\mathcal{F}_{\mathrm{DEAL}}$ box 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 aOTs from aBits with A as sender and B as receiver.

We go via a leaky version of authenticated OT, or LaOT, described in Fig. 14. The LaOT box 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 box aborts revealing that A is cheating. This means that if the box does not abort, with very high probability A only tried to guess a few choice bits.

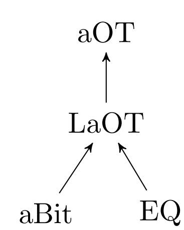

Fig. 13. Sect. 5 outline.

The protocol to construct a leaky aOT (described in Fig. 15) 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

Honest Parties: For $i = 1, \ldots, \ell$, the box 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 box in order to try to guess c_i. If $c_i \neq g_i$ the box will output `fail` and terminate. Otherwise the box 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 $\Delta = \Delta_A$.

Fig. 14. The Leaky Authenticated OT box LaOT(ℓ)

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 $[z]_{\mathsf{B}} = [r]_{\mathsf{B}} \oplus d$.

The protocol runs ℓ times in parallel, here described for a single leaky authenticated 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 $T_0, T_1 \in \{0,1\}^\kappa$.
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 *has both* (T_0, T_1).
9. A and B both input (T_0, T_1) to EQ. The comparisons are done using one call to EQ($\ell 2\kappa$).
10. If the values are the same, they output $[x_0]_{\mathsf{A}}, [x_1]_{\mathsf{A}}, [c]_{\mathsf{B}}, [z]_{\mathsf{B}}$.

Fig. 15. The protocol for authenticated OT with leaky choice bit

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 right MAC and x_1 together with a random string. Now if $c = 0$ B only sees the valid MAC and continues the protocol, while if $c = 1$ B aborts because of the wrong MAC. A similar attack can be mounted to check if $c = 1$. We will fix this later by randomly partitioning and combining a few LaOTs together.

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 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 EQ box 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. The proof of the following theorem can be found in App. F.

Theorem 5. *The protocol in Fig. 15 securely implements* $\mathrm{LaOT}(\ell)$ *in the* $(\mathrm{aBit}(4\ell, \kappa), \mathrm{EQ}(2\ell\kappa))$*-hybrid model.*

To deal with the leakage of the LaOT box, we let B randomly partition the LaOTs in small buckets: all the LaOTs in a bucket will be combined using an OT combiner (as shown in Fig. 16), in such a way that if at least one choice bit in every bucket is unknown to A, then the resulting aOT will not be leaky. The overall protocol is secure because of the OT combiner and the probability that any bucket is filled only with OTs where the choice bit leaked is negligible, as shown in App. G.

1. A and B generate $\ell' = B\ell$ authenticated OTs using $\mathrm{LaOT}(\ell')$. If the box 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. We describe how to combine two OTs from a bucket, call them $[x_0^1]_{\mathsf{A}}, [x_1^1]_{\mathsf{A}}, [c^1]_{\mathsf{B}}, [z^1]_{\mathsf{B}}$ and $[x_0^2]_{\mathsf{A}}, [x_1^2]_{\mathsf{A}}, [c^2]_{\mathsf{B}}, [z^2]_{\mathsf{B}}$. Call the result $[x_0]_{\mathsf{A}}, [x_1]_{\mathsf{A}}, [c]_{\mathsf{B}}, [z]_{\mathsf{B}}$. To combine more than two, just iterate by taking the result and combine it with the next leaky OT.
   (a) A reveals $d = x_0^1 \oplus x_1^1 \oplus x_0^2 \oplus x_1^2$.
   (b) Compute: $[c]_{\mathsf{B}} = [c^1]_{\mathsf{B}} \oplus [c^2]_{\mathsf{B}}$, $[z]_{\mathsf{B}} = [z^1]_{\mathsf{B}} \oplus [z^2]_{\mathsf{B}} \oplus d[c^1]_{\mathsf{B}}$, $[x_0]_{\mathsf{A}} = [x_0^1]_{\mathsf{A}} \oplus [x_0^2]_{\mathsf{A}}$, $[x_1]_{\mathsf{A}} = [x_0^1]_{\mathsf{A}} \oplus [x_1^2]_{\mathsf{A}}$.

Fig. 16. From Leaky Authenticated OTs to Authenticated OTs

Theorem 6. *Let* $\mathrm{aOT}(\ell)$ *denote the box which outputs* ℓ aOT*s as in* $\mathcal{F}_{\mathrm{DEAL}}$*. If* $(\log_2(\ell)+1)(B-1) \geq \psi$*, then the protocol in Fig. 16 securely implements* $\mathrm{aOT}(\ell)$ *in the* $\mathrm{LaOT}(B\ell)$*-hybrid model with security parameter* ψ*.*

6 Authenticated local AND

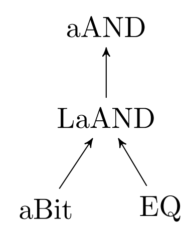

Fig. 17. Sect. 6 outline.

In this section we show how to generate aAND, i.e., how to implement the dealer box when it outputs $[x]_{\mathsf{A}}, [y]_{\mathsf{A}}, [z]_{\mathsf{A}}$ with $z = xy$. As usual, as aAND for B is symmetric, we only present how to construct aAND for A.

We first construct a leaky version of aAND, or LaAND, described in Fig. 18. Similar to the LaOT box the LaAND box may leak the value x to B, at the price for B of being detected. The intuition behind the protocol for LaAND, described in Fig. 19, 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 $E \neq 0^\kappa$. The returned value would be $H(W_0) \oplus xE$. To prevent this attack, they use the EQ box to compare the values instead. If B uses $E \neq 0^\kappa$, he must now guess x to pass the protocol. However, B still may use this technique to guess a few x bits. We fix this leakage later in a way similar to the way we fixed leakage of the LaOT box in Sect. 5. The proof of the following theorem can be found in App. H.

Theorem 7. *The protocol in Fig. 19 securely implements* LaAND(ℓ) *in the* (aBit($3\ell, \kappa$), EQ($\ell\kappa$))-*hybrid model.*

Honest Parties: For $i = 1, \ldots, \ell$, the box 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.
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 box in order to try to guess x_i. If $g_i \neq x_i$ the box will output `fail` to A and terminate. Otherwise the box 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 $\Delta = \Delta_A$.

Fig. 18. The box LaAND(ℓ) for ℓ Leaky Authenticated local AND.

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

1. A and B ask the dealer for $[x]_\mathsf{A}, [y]_\mathsf{A}, [r]_\mathsf{A}$. (The global key is Δ_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 EQ box, with inputs V and $H(K_x||K_z)$ respectively. All the ℓ calls to EQ are handled using a single call to EQ($\ell\kappa$).
7. If the strings were not different, the parties output $[x]_\mathsf{A}, [y]_\mathsf{A}, [z]_\mathsf{A}$.

Fig. 19. Protocol for authenticated local AND with leaking bit

We now handle a few guessed x bits by random bucketing and a straight-forward 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 LaAND(ℓ') with $\ell' = B\ell$. If the call to LaAND 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. The parties combine the B LaANDs in the same bucket. We describe how to combine two LaANDs, call them $[x^1]_\mathsf{A}, [y^1]_\mathsf{A}, [z^1]_\mathsf{A}$ and $[x^2]_\mathsf{A}, [y^2]_\mathsf{A}, [z^2]_\mathsf{A}$ into one, call the result $[x]_\mathsf{A}, [y]_\mathsf{A}, [z]_\mathsf{A}$:
   (a) A reveals $d = y^1 \oplus y^2$.
   (b) Compute $[x]_\mathsf{A} = [x^1]_\mathsf{A} \oplus [x^2]_\mathsf{A}$, $[y]_\mathsf{A} = [y^1]_\mathsf{A}$ and $[z]_\mathsf{A} = [z^1]_\mathsf{A} \oplus [z^2]_\mathsf{A} \oplus d[x^2]_\mathsf{A}$.
   To combine all B LaANDs in a bucket, just iterate by taking the result and combine it with the next element in the bucket.

Fig. 20. From Leaky Authenticated local ANDs to Authenticated local ANDs

Similar to the the way we removed leakage in Sect. 5 we start by producing $B\ell$ LaANDs. Then we randomly distribute the $B\ell$ LaANDs into ℓ buckets of size B. Finally we combine the LaANDs in each bucket into one aAND which is secure if at least one LaAND in the bucket was not leaky. The protocol is described in Fig. 20. The proof of Thm. 8 can be found in App. I.

Theorem 8. *Let* $\mathrm{aAND}(\ell)$ *denote the box which outputs* ℓ $\mathrm{aAND}s$ *as in* $\mathcal{F}_{\mathrm{DEAL}}$. *If* $(\log_2(\ell)+1)(B-1) \geq \psi$, *then the protocol in Fig. 20 securely implements* $\mathrm{aAND}(\ell)$ *in the* $\mathrm{LaAND}(B\ell)$*-hybrid model with security parameter* ψ.

This completes the description of our protocol. For the interested reader, a diagrammatic recap of the construction is given in App. J.

7 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 $\mathcal{F}_{\mathrm{DEAL}}$, 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 earlier of our prototype implementations which we discarded. Instead, in the current prototype, circuits are generated on the fly, in chunks which are 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 $\mathcal{F}_{\mathrm{DEAL}}$-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 $\mathcal{F}_{\mathrm{DEAL}}$-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.

In the timings we ran A and B on two different machines on Anonymous University's intranet (using two Intel Xeon E3430 2.40GHz cores on each machine). We recorded the number of Boolean

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

Fig. 21. 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.

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 Fig. 21 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 κ specifies that a poly-time adversary should have probability at most $\text{poly}(\kappa)2^{-\kappa}$ in breaking the protocol. The statistical security parameter σ specifies 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 κ, our keys and therefore MACs have length κ. 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.

References

AHI10. Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. Cryptology ePrint Archive, Report 2010/544, 2010. `http://eprint.iacr.org/`.

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.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

CHK+11. Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. Cryptology ePrint Archive, Report 2011/257, 2011. `http://eprint.iacr.org/`.

CKKZ11. Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the free-xor technique. Cryptology ePrint Archive, Report 2011/510, 2011. `http://eprint.iacr.org/`.

CvdGT95. Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *CRYPTO*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer, 1995.

DK10. Ivan Damgård and Marcel Keller. Secure multiparty aes. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 367–374. Springer, 2010.

DO10. Ivan Damgård and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In Tal Rabin, editor, *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 558–576. Springer, 2010.

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.

Gar04. Juan A. Garay. 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.

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.

Gol04. Oded Goldreich. *Foundations of Cryptography, Vol. 2.* Cambridge University Press, 2004. `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html`.

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.

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 *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 451–462, New York, NY, USA, 2010. ACM.

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.

IPS09. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 294–314. Springer, 2009.

JMN10. Thomas P. Jakobsen, Marc X. Makkes, and Janus Dam Nielsen. Efficient implementation of the orlandi protocol. In Jianying Zhou and Moti Yung, editors, *ACNS*, volume 6123 of *Lecture Notes in Computer Science*, pages 255–272, 2010.

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.

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.

LP11. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. TCC, 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.

MK10. Lior Malka and Jonathan Katz. VMCrypt - modular software architecture for scalable secure computation. Cryptology ePrint Archive, Report 2010/584, 2010. `http://eprint.iacr.org/`.

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/`.

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.

SS11. Abhi Shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In Paterson [Pat11], pages 386–405.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982.

A Complexity Analysis

We report here on the complexity analysis of our protocol. As showed in Corollary 1, the protocol requires an initial call to an ideal functionality for $(\mathrm{OT}(\frac{44}{3}\psi, \psi), \mathrm{EQ}(\psi))$. 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 EQ**:** The EQ box can be securely implemented with 2 calls to a hash function H.

Authenticated OT aOT**:** Every aOT costs $4B$ calls to aBit, $2B$ calls to EQ, and $6B$ calls to H, where B is the "bucket size".

Authenticated AND aAND**:** Every aAND costs $3B$ calls to aBit, B calls to 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 aOT, 2 aAND, 2 aBit.

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

The cost per aBit, in the protocol described in the paper, requires 59 calls to H. However, using some further optimizations (that are not described in the paper, 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 Sect. 7 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 Sect. 3 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.

B Proof of Thm. 1

The simulator can be built in a standard way, incorporating the $\mathcal{F}_{\mathrm{DEAL}}$ box 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 $\mathcal{F}_{\mathrm{DEAL}}$ to the corrupted parties allows the simulator to extract inputs used by corrupted parties and input these to the box $\mathcal{F}_{\mathrm{2PC}}$ on behalf of the corrupted parties. As an example, if A is corrupted, then learn the x_A sent to A by $\mathcal{F}_{\mathrm{DEAL}}$ in **Input** and observe the value x_B sent by A to B. Then input $x = x_A \oplus x_B$ to $\mathcal{F}_{\mathrm{2PC}}$. 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 $\mathcal{F}_{\mathrm{2PC}}$ 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 $\mathcal{F}_{\mathrm{DEAL}}$ 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 $\mathcal{F}_{\mathrm{DEAL}}$ 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 and its outputs from $\mathcal{F}_{\text{DEAL}}$.

It is easy to see that the protocol is passively secure and that if the adversary never sends an inconsistent bit, then it is perfectly following the protocol up to input substitution. So, to prove security it is enough to prove that the adversary manages to send an inconsistent bit with negligible probability. However, sending an inconsistent bit turns out to be equivalent to guessing the global key Δ.

We now formalize the last claim. Consider the following game $\Game_{I,I}$ played by an attacker A:

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

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

Break query I: If A outputs a query $(\mathtt{break}, a_1, l_1, \ldots, a_p, l_p, M')$, 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 $\Game_{II,II}$ played by an attacker A:

Global key: *No change.*

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

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

We let $\Game_{II,I}$ 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 1. *For any adversary $\mathsf{A}_{I,I}$ for $\Game_{I,I}$ there exists an adversary $\mathsf{A}_{II,I}$ for $\Game_{II,I}$ 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 $\Game_{I,I}$, consider the following adversary $\mathsf{A}_{II,I}$ for $\Game_{II,I}$. The adversary $\mathsf{A}_{II,I}$ passes all side information on Δ to $\mathsf{A}_{I,I}$. If $\mathsf{A}_{I,I}$ outputs $(\mathtt{mac}, b, l)$, then $\mathsf{A}_{II,I}$ samples $M \in_{\mathrm{R}} \{0,1\}^\kappa$, outputs $(\mathtt{mac}, b, l, M)$ to $\Game_{II,I}$ 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 $\Game_{II,I}$. 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 $\Game_{I,I}$ the value K is uniform and $M = K \oplus b\Delta$. In $\Game_{II,I}$ the value M is uniform and $K = M \oplus b\Delta$. This gives the exact same distribution on (K, M). □

Lemma 2. *For any adversary $\mathsf{A}_{II,I}$ for $\Game_{II,I}$ there exists an adversary $\mathsf{A}_{II,II}$ for $\Game_{II,II}$ 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 $\eth_{II,I}$, consider the following adversary $\mathsf{A}_{II,II}$ for $\eth_{II,II}$. The adversary $\mathsf{A}_{II,II}$ passes any side information on $\varDelta$ to $\mathsf{A}_{II,I}$. If $\mathsf{A}_{II,I}$ outputs $(\texttt{mac}, b, l, M)$, then $\mathsf{A}_{II,II}$ outputs $(\texttt{mac}, b, l, M)$ to $\eth_{II,II}$ and stores (l, M, b). If $\mathsf{A}_{II,I}$ outputs $(\texttt{break}, a_1, l_1, \ldots, a_p, l_p, M')$, 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 $(\texttt{break}, M \oplus M')$. For each (l_i, M_i, b_i) let K_i be the corresponding key stored by $\eth_{II,II}$. We have that $M_i = K_i \oplus b_i \oplus \varDelta$, so if we let $K = \oplus_{i=1}^{p} a_i K_i$, then $M = K \oplus b\varDelta$. Assume that $\mathsf{A}_{II,I}$ would win $\eth_{II,I}$, i.e., $M' = K \oplus (1 \oplus b)\varDelta$. This implies that $M \oplus M' = K \oplus b\varDelta \oplus K \oplus (1 \oplus b)\varDelta = \varDelta$, which means that $\mathsf{A}_{II,II}$ wins $\eth_{II,II}$. □

Consider then the following game $\eth_{II}$ played by an attacker A:

Global key: *No change.*
MAC query: *No MAC queries are allowed.*
Break query II: *No change.*

Lemma 3. *For any adversary $\mathsf{A}_{II,II}$ for $\eth_{II,II}$ there exists an adversary A_{II} for $\eth_{II}$ 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 $\eth_{II}$ 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. □

Corollary 2. *For any adversary $\mathsf{A}_{I,I}$ for $\eth_{I,I}$ there exists an adversary A_{II} for $\eth_{II}$ which is no stronger than $\mathsf{A}_{I,I}$ and which wins the game with the same probability as $\mathsf{A}_{I,I}$.*

This formalizes the claim that the only way to break the scheme is to guess $\varDelta$.

C Proof of Thm. 2

The simulator answers a global key query $\varGamma$ to WaBit by doing the global key query $\mathbf{A}\varGamma$ on the ideal functionality aBit and returning the reply. This gives a perfect simulation of these queries, and we ignore them below.

Correctness of the protocol is straightforward: We have that $M'_i = K'_i \oplus x_i \varGamma_A$, so $M_i = \mathbf{A}M'_i = \mathbf{A}K'_i \oplus x_i \mathbf{A}\varGamma_A = K_i \oplus x_i \varDelta_A$. Clearly the protocol leaks no information on the x_i's as there is only communication from B to A. It is therefore sufficient to look at the case where A is corrupted. We are not going to give a simulation argument but just show that $\varDelta_A$ is uniformly random in the view of A except with probability $2^{2-\psi}$. Turning this argument into a simulation argument is straight forward.

We start by proving three technical lemmas.

Assume that $\mathcal{L}$ is a class of leakage functions on τ bits which is κ-secure. Consider the following game.

1. Sample $\varGamma_A \in_{\mathrm{R}} \{0,1\}^{\tau}$.
2. Get $L \in \mathcal{L}$ from A and sample $(S, c) \leftarrow L$.
3. Give $\{(j, (\varGamma_A)_j)\}_{j \in S}$ to A.
4. Sample $\mathbf{A} \in_{\mathrm{R}} \{0,1\}^{\psi \times \tau}$ and give $\mathbf{A}$ to A.
5. Let $\varDelta_A = \mathbf{A}\varGamma_A$.

We want to show that Δ_A is uniform to A except with probability $2^{2-\psi}$. When we say that Δ_A *is uniform to* A we mean that Δ_A 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 $2^{2-\psi}$ and
2. when F does not occur, then Δ_A is uniform to A.

For a subset $S \subset \{1,\ldots,\tau\}$ of the column indices, let $\mathbf{A}^S$ be the matrix where column j is equal to $\mathbf{A}^j$ 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 $\boldsymbol{v}$ we use the notation $\boldsymbol{v}_S$ to mean that we set all indices v_i where $i \notin S$ to be 0. Note that $\mathbf{A}\boldsymbol{v}_S = \mathbf{A}^S\boldsymbol{v}$. Let $\overline{S} = \{1,\ldots,\tau\} \setminus S$.

Lemma 4. *Let S be the indices of the bits learned by A and let $\boldsymbol{A}$ be the matrix in the game above. If $\boldsymbol{A}^{\overline{S}}$ spans $\{0,1\}^\psi$, then Δ_A is uniform to A.*

Proof. We start by making two simple observations. First of all, if A learns $(\Gamma_A)_j$ for $j \in S$, then it learns $(\Gamma_A)_S$[5], so it knows $\mathbf{A}(\Gamma_A)_S = \mathbf{A}^S\Gamma_A$. The second observation is that $\mathbf{A}\Gamma_A = \mathbf{A}^S\Gamma_A + \mathbf{A}^{\overline{S}}\Gamma_A$, as $\mathbf{A} = \mathbf{A}^S + \mathbf{A}^{\overline{S}}$. The lemma follows directly from these observations and the premise: We have that $\mathbf{A}^{\overline{S}}\Gamma_A$ is uniformly random in $\{0,1\}^\psi$ when the columns of $\mathbf{A}^{\overline{S}}$ span $\{0,1\}^\psi$. Since $\mathbf{A}^{\overline{S}}\Gamma_A = \mathbf{A}(\Gamma_A)_{\overline{S}}$ and $(\Gamma_A)_{\overline{S}}$ is uniformly random and independent of the view of A it follows that $\mathbf{A}^{\overline{S}}\Gamma_A$ is uniformly random and independent of the view of A. Since $\mathbf{A}^S\Gamma_A$ is known by A it follows that $\mathbf{A}^S\Gamma_A + \mathbf{A}^{\overline{S}}\Gamma_A$ is uniform to A. The proof concludes by using that $\Delta_A = \mathbf{A}^S\Gamma_A + \mathbf{A}^{\overline{S}}\Gamma_A$. □

Lemma 5. *Let W be the event that $|S| \geq \tau - n$ and $c = 1$. Then* $\Pr[W] \leq 2^{-\psi}$.

Proof. We use that

- $\kappa = \frac{3}{4}\tau$,
- $\tau = \alpha n$ for $\alpha = \frac{44}{27}$,
- $n = \frac{9}{2}\psi$,
- $\mathcal{L}$ is κ-secure on τ bits.

Without loss of generality we can assume that A plays an optimal $L \in \mathcal{L}$, i.e., $\log_2(\mathrm{E}\left[c2^{|S|}\right]) = \mathrm{leak}_{\mathcal{L}}$. Since $\mathcal{L}$ is κ secure on τ bits, it follows that $\mathrm{leak}_{\mathcal{L}} \leq \tau - \kappa = \frac{1}{4}\tau$. This gives that

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

which we use later.

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

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

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

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

[5] Here we are looking at the string Γ_A as a column vector of bits.

Hence

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

Combining with (1) 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 checked to be the case by definition of τ, α, n and ψ. □

Lemma 6. *Let $x_1, \dots, x_n \in_{\mathrm{R}} \{0,1\}^\psi$. Then $x_1, \dots, x_n$ span $\{0,1\}^\psi$ except with probability $2^{1-\psi}$.*

Proof. We only use that

– $n = \frac{9}{2}\psi$.

Define random variables $Y_1, \dots, Y_n$ where $Y_i = 0$ if $x_1, \dots, x_{i-1}$ spans $\{0,1\}^\psi$ or the span of $x_1, \dots, x_{i-1}$ does not include x_i. Let $Y_i = 1$ in all other cases. Note that if $x_1, \dots, x_{i-1}$ spans $\{0,1\}^\psi$, then $\Pr[Y_i = 1] = 0 \leq \frac{1}{2}$ and that if $x_1, \dots, x_{i-1}$ 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

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

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 $x_1, \dots, x_n$ contains at least ψ linear independent vectors. □

Recall that W is the event that $|S| \geq \tau - n$ and $c = 1$. By Lemma 5 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., $|S| \leq \tau - n$ and hence $|\overline{S}| \geq \tau = \frac{9}{2}\psi$. Since $\mathbf{A}$ is picked uniformly at random and independent of S it follows that $\frac{9}{2}\psi$ of the columns in $\mathbf{A}^{\overline{S}}$ are uniformly random and independent. Hence, by Lemma 6, 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 4 Δ_A is uniform to A. I.e., if the event $F = W \cup D$ does not happen, then Δ_A is uniform to A. And, $\Pr[F] \leq \Pr[W] + \Pr[D] \leq 2^{-\psi} + 2^{1-\psi} \leq 2^{2-\psi}$.

D Proof of Thm. 4

Notice that since we have to prove that we implement LaBit, which has the global key queries, it would be stronger to show that we implement a version of LaBit′ which does not have these global key queries. This is what we do below, as we let LaBit denote this stronger box.

Given a pairing π, let $\mathcal{S}(\pi) = \{i | i < \pi(i)\}$, i.e., for each pair we add the smallest indexed to $\mathcal{S}(\pi)$.

The cases where no party is corrupted and where B is corrupted is straight forward, so we will focus on the case that A is corrupted.

The proof goes via a number of intermediary boxes, and for each we show linear reducibility.

Approximating LaBit, Version 1 This box captures the fact that the only thing a malicious A can manage is to use different Γ's in a few bit authentications.

Honest-Parties: As in LaBit.
Corrupted Parties:

1. If B is corrupted: As in LaBit.
2. (a) If A is corrupted, then A inputs a functions $\mathrm{col} : \{1, \dots, \mathcal{T}\} \to \{1, \dots, \mathcal{T}\}$. We think of col as assigning colors from $\{1, \dots, \mathcal{T}\}$ to $\mathcal{T}$ balls named $1, \dots, \mathcal{T}$. In addition A inputs $\Lambda_1, \dots, \Lambda_{\mathcal{T}} \in \{0,1\}^\ell$ and $L_1, \dots, L_{\mathcal{T}} \in \{0,1\}^\ell$.
   (b) Then the box samples a uniformly random pairing $\pi : \{1, \dots, \mathcal{T}\} \to \{1, \dots, \mathcal{T}\}$ and outputs π to A. We think of π as pairing the $\mathcal{T}$ balls. Let $\mathcal{S} = \mathcal{S}(\pi)$ and let $\mathcal{M} = \{i \in \mathcal{S} | \mathrm{col}(i) \neq \mathrm{col}(\pi(i))\}$. We call $i \in \mathcal{M}$ a *mismatched* ball.
   (c) Now A inputs the guesses $\{(i, g_i)\}_{i \in \mathcal{M}}$.
   (d) The box samples $(y_1, \dots, y_{\mathcal{T}}) \in_{\mathrm{R}} \{0,1\}^{\mathcal{T}}$. Then the box lets $c = 1$ if $g_i = y_i$ for $i \in \mathcal{M}$, otherwise it lets $c = 0$. If $c = 0$ the box outputs `fail` to B and terminates. Otherwise, for $i \in \mathcal{S}$ it computes $N_i = L_i \oplus y_i \Lambda_{\mathrm{col}(i)}$ and outputs $\{((N_i, y_i)\}_{i \in \mathcal{S}}$ to B.

Fig. 22. The First Intermediate Box IB1

Lemma 7. IB1 *is linear reducible to* $(\mathrm{OT}(2\tau, \ell), \mathrm{EQ}(\tau\ell))$.

Proof. By observing A's inputs to the OTs, the simulator learns all $(Y_{i,0}, Y_{i,1})$. Let $L_i = Y_{i,0}$ and $\Gamma_i = Y_{i,0} \oplus Y_{i,1}$.

Let $f = |\{\Gamma_i\}_{i=1}^{\mathcal{T}}|$ and pick distinct $\Lambda_1, \dots, \Lambda_f$ and $\mathrm{col} : \{1, \dots, \mathcal{T}\} \to \{1, \dots, \mathcal{T}\}$ such that $\Gamma_i = \Lambda_{\mathrm{col}(i)}$. By construction

$$
\begin{aligned}
Y_{i,1} &= Y_{i,0} \oplus (Y_{i,0} \oplus Y_{i,1}) \\
&= L_i \oplus \Gamma_i \\
&= L_i \oplus \Lambda_{\mathrm{col}(i)} \ .
\end{aligned}
$$

Input col and $\Lambda_1, \dots, \Lambda_f$ and $L_1, \dots, L_{\mathcal{T}}$ to IB1 on behalf of A and receive π. Send π to A as if coming from B along with uniformly random $\{d_i\}_{i \in \mathcal{S}}$.

Then observe the inputs Z_i from A to the EQ box.

The simulator must now pick the guesses g_i for $i \in \mathcal{M}$. Note that $i \in \mathcal{M}$ implies that $\Lambda_{\mathrm{col}(i)} \neq \Lambda_{\mathrm{col}(\pi(i))}$, which implies that $\Gamma_i \neq \Gamma_{\pi(i)}$. We use this to pick g_i, as follows: after seeing d_i, A knows that either $(y_i, y_{\pi(i)}) = (0, d_i)$ or $(y_i, y_{\pi(i)}) = (1, 1 \oplus d_i)$. Hence an honest B would input to the comparison the following value depending on y_i

$$
W_i(y_i) = (L_i \oplus L_{\pi(i)} \oplus d_i \Lambda_{\mathrm{col}(\pi(i))}) \oplus y_i (\Lambda_{\mathrm{col}(i)} \oplus \Lambda_{\mathrm{col}(\pi(i))}) \ .
$$

As $i \in \mathcal{M}$, the mismatched set, $\Lambda_{\mathrm{col}(i)} \neq \Lambda_{\mathrm{col}(\pi(i))}$ and therefore $W_i(0) \neq W_i(1)$. Therefore if A's input to the EQ box Z_i is equal to $W_i(0)$ (resp. $W_i(1)$), the simulator inputs a guess $g_i = 0$ (resp. $g_i = 1$). In any other case, the simulator outputs `fail` and aborts.

Notice that in the real-life protocol, if $g_i = y_i$, then $N_i = W_i(y_i) = Z_i$ and A passes the test. If $g_i \neq y_i$, then $N_i = W_i(1 \oplus c_i) \neq Z_i$ and A fails the test. So, the protocol and the simulation fails on the same event. Note then that when the box does not fail, then it outputs

$$\begin{aligned} N_i &= L_i \oplus y_i \Lambda_{\mathrm{col}(i)} \\ &= Y_{i,0} \oplus y_i \Gamma_i \\ &= Y_{i,0} \oplus y_i (Y_{i,0} \oplus Y_{i,1}) \\ &= Y_{i,y_i} \ , \end{aligned}$$

exactly as the protocol. Hence the simulation is perfect. □

Approximating LaBit, Version 2 We now formalize the idea that a wrong Γ-value is no worse that a leaked bit.

We first need a preliminary definition of the most common color called col_0. If several colors are most common, then arbitrarily pick the numerically largest one. To be more precise, for each color c, let $C(c) = \{j \in \{1, \dots, \mathcal{T}\} | \, \mathrm{col}(j) = c\}$, let $a_0 = \max_c |C(c)|$ and let $\mathrm{col}_0 = \max\{c | C(c) = a_0|\}$.

Consider the following box IB2 in Fig. 23 for formalizing the second idea.

Honest-Parties: As in LaBit.
Corrupted Parties:

1. If B is corrupted: As in LaBit.
2. (a) If A is corrupted, then A inputs a function $\mathrm{col} : \{1, \dots, \mathcal{T}\} \to \{1, \dots, \mathcal{T}\}$.
   (b) Then the box samples a uniformly random pairing $\pi : \{1, \dots, \mathcal{T}\} \to \{1, \dots, \mathcal{T}\}$ and outputs π to A. Let $\mathcal{S} = \mathcal{S}(\pi)$ and $\mathcal{M} = \{i \in \mathcal{S} | \, \mathrm{col}(i) \neq \mathrm{col}(\pi(i))\}$.
   (c) Now A inputs the guesses $\{(i, g_i)\}_{i \in \mathcal{M}}$.
   (d) The box lets $c = 1$ if $g_i = y_i$ for $i \in \mathcal{M}$, otherwise it lets $c = 0$. If $c = 0$ the box outputs `fail` to A and terminates. Otherwise, the box determines col_0.
   Then for $i \in \mathcal{S}$, if $\mathrm{col}(i) \neq \mathrm{col}_0$, the box outputs (i, y_i) to A. Then A inputs $L_1, \dots, L_{\mathcal{T}} \in \{0,1\}^\ell$ and $\Gamma_B \in \{0,1\}^\ell$ and for $i \in \mathcal{S}$ the box computes $N_i = L_i \oplus y_i \Gamma_B$. Then it outputs $\{(N_i, y_i)\}_{i \in \mathcal{S}}$ to B.

Fig. 23. The Second Intermediate Box IB2

Lemma 8. IB2 *is linear locally reducible to* IB1.

Proof. The implementation of IB2 consist simply of calling IB1.

The case where B or no party is corrupted is trivial, so assume that A is corrupted. Note that the simulator must simulate IB2 to the environment and is the one simulating IB1 to the corrupted A.

First the simulator observes the inputs col, $\Lambda_1, \dots, \Lambda_{\mathcal{T}} \in \{0,1\}^\ell$ and $L_1, \dots, L_{\mathcal{T}} \in \{0,1\}^\ell$ of A^* to IB1 and inputs col to IB2.

Then IB2 outputs π and the simulator inputs π to A^* as if coming from IB1, and computes $\mathcal{M}$ as IB1 and IB2 would have done.

Then the simulator observes the guesses $\{(i, g_i)\}_{i \in \mathcal{M}}$ from A^* to IB1 and inputs $\{(i, g_i)\}_{i \in \mathcal{M}}$ to IB2. If IB2 outputs `fail` to B the simulation is over, and it is perfect as IB1 and IB2 fail based on the same event. If IB2 does not fail it determines col_0 and for $i \in \mathcal{M}$, if $\mathrm{col}(i) \neq \mathrm{col}_0$, the box outputs (i, y_i) to the simulator. The simulator can also determine col_0.

Now let $\Gamma_B = \Lambda_{\mathrm{col}_0}$ and for $i \in \mathcal{M}$, if $\mathrm{col}(i) = \mathrm{col}_0$, let $L'_i = L_i$. Then for $i \in \mathcal{M}$, if $\mathrm{col}(i) \neq \mathrm{col}_0$, let $L'_i = (L_i \oplus y_i \Lambda_{\mathrm{col}(i)}) \oplus y_i \Gamma_B$. Then input $L'_i, \ldots, L'_{\mathcal{T}}$ and Γ_B to IB2.

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

Approximate LaBit, Version 3 We now massage IB2 a bit to make it look like LaBit. As a step towards this, consider the box IB3 in Fig. 24.

Honest-Parties: As in LaBit.
Corrupted Parties:

1. Corrupted B: As in LaBit.
2. (a) If A is corrupted, then A inputs a function $\mathrm{col} : \{1, \ldots, \mathcal{T}\} \to \{1, \ldots, \mathcal{T}\}$.
   (b) Then the box samples a uniformly random pairing $\pi : \{1, \ldots, \mathcal{T}\} \to \{1, \ldots, \mathcal{T}\}$ and outputs π to A. Let $\mathcal{M} = \{i \in \mathcal{S} | \mathrm{col}(i) \neq \mathrm{col}(\pi(i))\}$. The box flips a coin $c \in \{0, 1\}$ with $c = 1$ with probability $2^{-|\mathcal{M}|}$. If $c = 0$ the box outputs `fail` to B and terminates. Otherwise, the box outputs `success` and the game proceeds.
   (c) Now A inputs the guesses $\{(i, g_i)\}_{i \in \mathcal{M}}$.
   (d) The box updates $y_i \leftarrow g_i$ for $i \in \mathcal{M}$. Then the box determines col_0. Then for $i = \mathcal{S} \setminus \mathcal{M}$, if $\mathrm{col}(i) \neq \mathrm{col}_0$, the box outputs i to A who inputs $g_i \in \{0, 1\}$ and the box updates $y_i \leftarrow g_i$.
   (e) Then A inputs $L_1, \ldots, L_{\mathcal{T}} \in \{0, 1\}^{\ell}$ and $\Gamma_B \in \{0, 1\}^{\ell}$ and for $i \in \mathcal{S}$ the box computes $N_i = L_i \oplus y_i \Gamma_B$. Then it outputs $\{(N_i, y_i)\}_{i \in \mathcal{S}}$ to B.

Fig. 24. The third Intermediate Box, IB3

Lemma 9. IB3 *is linear locally reducible to* IB2*.*

Proof. It is easy to see that IB3 is linear locally reducible to IB2—again the implementation consist simply of calling IB2. To see this, consider first the change in how the box fails and how the y_i for $i \in \mathcal{M}$ are set. In IB2 the box fails exactly with probability $2^{-|\mathcal{M}|}$ as the probability that $g_i = y_i$ for $i \in \mathcal{M}$ is exactly $2^{-|\mathcal{M}|}$. Furthermore, if IB2 does not fail, then $y_i = g_i$ for $i \in \mathcal{M}$. So, this is exactly the same behavior as IB3, hence this change is really just another way to implement the same box. As for the second change, the simulator will input a uniformly random $g_i \in_{\mathrm{R}} \{0, 1\}$ to IB3 when IB3 outputs i and will then show (i, y_i) to the corrupted A^* expecting to interact with IB2. □

We then argue that we can define a class $\mathcal{L}$ such that $\mathrm{LaBit}^{\mathcal{L}}$ is linear locally reducible to IB3. Let $\mathcal{L}$ be the following class.

- A leakage function is specified by $L = \mathrm{col}$, where $\mathrm{col} : \{1, \ldots, \mathcal{T}\} \to \{1, \ldots, \mathcal{T}\}$.
- To sample a leakage function $L = \mathrm{col}$, sample a uniformly random pairing $\pi : \{1, \ldots, \mathcal{T}\} \to \{1, \ldots, \mathcal{T}\}$, let $\mathcal{S} = \mathcal{S}(\pi)$, let $\Pi : \mathcal{S}(\pi) \to \{1, \ldots, \tau\}$ be the order preserving permutation, let $\mathcal{M} = \{j \in \mathcal{S} | \mathrm{col}(j) \neq \mathrm{col}(\pi(j))\}$, let $c = 1$ with probability $2^{-|\mathcal{M}|}$ and $c = 0$ otherwise, let col_0 be the most common color as defined before, let $S' = \mathcal{M} \cup \{j \in \mathcal{S} | \mathrm{col}(j) \neq \mathrm{col}_0\}$, $S = \pi(S')$ and output (c, S).

Playing with IB3 and $\mathrm{LaBit}^{\mathcal{L}}$ will give the same failure probability and will allow to specify the same bits. The only difference is that when playing with $\mathrm{LaBit}^{\mathcal{L}}$, the corrupted A^* does not get to see π, as $\mathrm{LaBit}^{\mathcal{L}}$ does not leak the randomness used to sample the leakage function L. Below we argue that given c and S one can efficiently sample a uniformly random pairing π which would lead to S given c. Turning this into a simulation argument is easy: the simulator will know c and S and will sample π from these and show this π to A^*, hence perfectly simulating IB3. This gives the following lemma.

Lemma 10. $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$ *is linear locally reducible to* IB3.

The simulator knows col and S and it can determine col_0. From col_0 the simulator can also compute

$$\begin{aligned} T &= S \setminus \{j \in \{1, \ldots, \mathcal{T}\} | \operatorname{col}(j) \neq \mathrm{col}_0\} \\ &= \mathcal{M} \cap \{j \in \{1, \ldots, \mathcal{T}\} | \operatorname{col}(j) = \mathrm{col}_0\} \\ &= \{j \in \{1, \ldots, \mathcal{T}\} | \operatorname{col}(j) = \mathrm{col}_0 \wedge \operatorname{col}(j) \neq \operatorname{col}(\pi(j))\} \\ &= \{j \in \{1, \ldots, \mathcal{T}\} | \operatorname{col}(j) = \mathrm{col}_0 \wedge \operatorname{col}(\pi(j)) \neq \mathrm{col}_0\} \ . \end{aligned}$$

This restriction is meet iff π has the property that $\operatorname{col}(\pi(j)) \neq \mathrm{col}_0$ for $j \in T$ and $\operatorname{col}(\pi(j)) = \mathrm{col}_0$ for $j \in C_0 \setminus T$, where $C_0 = \{j | \operatorname{col}(j) = \mathrm{col}_0\}$. Furthermore, any π meeting this restrictions would lead to the observed value of π. It is hence sufficient to show that we can sample a uniformly random π meeting these restrictions.

Let $\overline{C_0} = \{1, \ldots, \mathcal{T}\} \setminus C_0$. Pick $\pi_0 : T \to \overline{C_0}$ to be a uniformly random injection on the specified domains. Pick $\pi_1 : C_0 \setminus T \to C_0$ similarly. Let $\pi_2 : T \cup C_0 \to \{1, \ldots, \tau\}$ be defined by $\pi_2(j) = \pi_0(j)$ for $j \in T$ and $\pi_2(j) = \pi_1(j)$ for $j \in C_0 \cup T$. Since π_0 and π_1 map into disjoint sets, this is again an injection. Now let $\pi_3 : \{1, \ldots, \tau\} \setminus (C_0 \cup T) \to \{1, \ldots, \tau\} \setminus \operatorname{img}(\pi_2)$ be a random permutation on the specified domains. Define π from π_2 and π_3 as we defined π_2 from π_0 and π_1. Then it is easy to see that π is a uniformly random permutation meeting the restrictions. The definition of π shows how to sample it efficiently.

Concluding the Proof Using the above theorem and lemmata and the fact that linear reducibility is transitive, we now have the following theorem.

Corollary 3. $\mathrm{LaBit}^{\mathcal{L}}(\tau, \ell)$ *is linear reducible to* $(\mathrm{OT}(2\tau, \ell), \mathrm{EQ}(\tau\ell))$.

We now show that if we set $\kappa = \frac{3}{4}\tau$, then $\mathcal{L}$ is κ-secure. For this purpose we assign a price to each ball $j \in \mathcal{S}(\pi)$.

1. If $\operatorname{col}(j) \neq \operatorname{col}(\pi(j))$, then let $\mathrm{price}_{\mathrm{col},\pi}(j) = 1$.
2. If $\operatorname{col}(j) = \operatorname{col}(\pi(j)) = \mathrm{col}_0$, then let $\mathrm{price}_{\mathrm{col},\pi}(j) = 1$.
3. If $\operatorname{col}(j) = \operatorname{col}(\pi(j)) \neq \mathrm{col}_0$, then let $\mathrm{price}_{\mathrm{col},\pi}(j) = 0$.

Let $\mathrm{price}_{\mathrm{col},\pi} = \sum_{j \in \mathcal{S}} \mathrm{price}_{\mathrm{col},\pi}(j)$.

Lemma 11. *Consider an adversary* A *playing the game against* $\mathcal{L}$ *and assume that it submits* $L = \mathrm{col}$. *Assume that the game uses* π. *Then the success probability of* A *is at most* $2^{-\mathrm{price}_{\mathrm{col},\pi}}$.

Proof. Define $\mathrm{price}^1_{\mathrm{col},\pi}(j)$ as $\mathrm{price}_{\mathrm{col},\pi}(j)$ except that if $\operatorname{col}(j) = \operatorname{col}(\pi(j)) = \mathrm{col}_0$, then $\mathrm{price}^1_{\mathrm{col},\pi}(j) = 0$. Define $\mathrm{price}^2_{\mathrm{col},\pi}(j)$ as $\mathrm{price}_{\mathrm{col},\pi}(j)$ except that if $\operatorname{col}(j) \neq \operatorname{col}(\pi(j))$, then $\mathrm{price}_{\mathrm{col},\pi}(j) = 0$. Then $\mathrm{price}_{\mathrm{col},\pi}(j) = \mathrm{price}^1_{\mathrm{col},\pi}(j) + \mathrm{price}^2_{\mathrm{col},\pi}(j)$. Define $\mathrm{price}^1_{\mathrm{col},\pi}$ and $\mathrm{price}^2_{\mathrm{col},\pi}$ by summing over $j \in \mathcal{S}$. Then $\mathrm{price}_{\mathrm{col},\pi} = \mathrm{price}^1_{\mathrm{col},\pi} + \mathrm{price}^2_{\mathrm{col},\pi}$. Note that $|\mathcal{M}| = \mathrm{price}_{\mathrm{col},\pi}(j)$ and note that $|S'| = \tau - \mathrm{price}^2_{\mathrm{col},\pi}(j)$,[6] as the only balls $j \in \mathcal{S}$ which do not enter S' are

[6] Recall that S' is defined during the definition of $\mathcal{L}$ above.

those for which $\mathrm{col}(j) = \mathrm{col}(\pi(j)) = \mathrm{col}_0$. We have that A wins if $c = 1$ and he guesses $y_{\Pi(j)}$ for $j \in \mathcal{S} \setminus S'$. The probability that $c = 1$ is $2^{-|\mathcal{M}|} = 2^{-\mathrm{price}_{\mathrm{col},\pi}(j)}$. We have that $|\mathcal{S} \setminus S'| = |\mathcal{S}| - |S'| = \tau - (\tau - \mathrm{price}^2_{\mathrm{col},\pi}(j)) = \mathrm{price}^2_{\mathrm{col},\pi}(j)$. So, the probability that A guesses correctly is $2^{-\mathrm{price}^2_{\mathrm{col},\pi}(j)}$. So, the overall success probability is $2^{-\mathrm{price}^1_{\mathrm{col},\pi}(j)} 2^{-\mathrm{price}^2_{\mathrm{col},\pi}(j)} = 2^{-\mathrm{price}_{\mathrm{col},\pi}(j)}$. $\square$

Now let π be chosen uniformly at random and let $\mathrm{price}_{\mathrm{col}(j)}$ be the random variable describing $\mathrm{price}_{\mathrm{col},\pi}(j)$. Let $\mathrm{price}_{\mathrm{col}} = \sum_{j \in \mathcal{S}} \mathrm{price}_{\mathrm{col}(j)}$. It is then easy to see that the probability of winning the game on $L = \mathrm{col}$ is at most

$$\mathrm{success}_{\mathrm{col}} = \sum_{p=0}^{\tau} \Pr\left[\mathrm{price}_{\mathrm{col}} = p\right] 2^{-p} \ .$$

For each price p, let P_p be an index variable which is 1 if $\mathrm{price}_{\mathrm{col}} = p$ and which is 0 otherwise. Note that $\mathrm{E}\left[P_p\right] = \Pr\left[\mathrm{price}_{\mathrm{col}} = p\right]$, and note that $\sum_{p=0}^{\tau} P_p 2^{-p} = 2^{-\mathrm{price}_{\mathrm{col}}}$ as $P_p = 0$ for $p \neq \mathrm{price}_{\mathrm{col}}$ and $P_p = 1$ for $p = \mathrm{price}_{\mathrm{col}}$. Then

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

Now let $\phi(x) = 2^{-x}$, and we have that

$$\mathrm{success}_{\mathrm{col}} = \mathrm{E}\left[\phi(\sum_{j \in \mathcal{S}} \mathrm{price}_{\mathrm{col}}(j))\right] \ .$$

Since $\phi(x)$ is concave it follows from Jensen's inequality that

$$\mathrm{E}\left[\phi(\sum_{j \in \mathcal{S}} \mathrm{price}_{\mathrm{col}}(j))\right] \leq \phi\left(\mathrm{E}\left[\sum_{j \in \mathcal{S}} \mathrm{price}_{\mathrm{col}}(j)\right]\right) \ .$$

Hence

$$\mathrm{success}_{\mathrm{col}} \leq 2^{-\mathrm{E}\left[\sum_{j=1}^{\tau} \mathrm{price}_{\mathrm{col}}(\Pi^{-1}(j))\right]} = 2^{-\sum_{j=1}^{\tau} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(\Pi^{-1}(j))\right]} = 2^{-\sum_{j \in \mathcal{S}} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right]} \ .$$

It follows that if we can compute $m_0 = \min_{\mathrm{col}} \sum_{j \in \mathcal{S}} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right]$, then 2^{-m_0} is an upper bound on the best success rate.

We say that $L = \mathrm{col}$ is optimal if $\sum_{j \in \mathcal{S}} \mathrm{E}\left[\mathrm{price}_L(j)\right] = m_0$, and now find an optimal L.

We first show that there is no reason to use balls of color col_0 in the optimal strategy.

Lemma 12. *Let* $L = \mathrm{col}$ *be an optimal leakage function and let* $\mathrm{col}_0 = \mathrm{col}_0(\mathrm{col})$*. Then there exist* j *such that* $\mathrm{col}(j) \neq \mathrm{col}_0$*.*

Proof. Assume for the sake of contradiction that $\mathrm{col}(j) = \mathrm{col}_0$ for $j = 1, \dots, \mathcal{T}$. Then clearly $\sum_{j \in \mathcal{S}} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] = \tau$, and it is easy to see that there are strategies which do better than $2^{-\tau}$, so L cannot be optimal. $\square$

Let $\mathrm{col}_1, \dots, \mathrm{col}_{\mathcal{T}}$ be an enumeration of the colors different from col_0, i.e., $\{\mathrm{col}_0, \mathrm{col}_1, \dots, \mathrm{col}_{\mathcal{T}}\} = \{1, \dots, \mathcal{T}\}$. Let C_i be the balls with color col_i, i.e., $C_i = \{j \in \{1, \dots, \mathcal{T}\} | \mathrm{col}(j) = \mathrm{col}_i\}$. Note that $\{1, \dots, \mathcal{T}\}$ is a disjoint union of $C_1, \dots, C_{\mathcal{T}}$. Let a_i be the number of balls of color col_i, i.e., $a_i = |C_i|$. Note that $\mathcal{T} = \sum_{i=1}^{\mathcal{T}} a_i$.

With these definitions we have that

$$\sum_{j=1}^{\mathcal{T}} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] = \sum_{i=1}^{\mathcal{T}} \sum_{j \in C_i} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] \ .$$

For a ball $j \in C_0$ of color col_0 we always have $\mathrm{price}_{\mathrm{col}}(j) = \frac{1}{2}$, by definition of the price, so

$$\sum_{j \in C_0} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] = \sum_{j \in C_0} \frac{1}{2} = \frac{1}{2} a_0 \ .$$

For a ball $j \in C_i$ for $i > 0$ we have $\mathrm{price}_{\mathrm{col}}(j) = 0$ if $\mathrm{col}(\pi(j)) = \mathrm{col}_i$ and $\mathrm{price}_{\mathrm{col}}(j) = \frac{1}{2}$ if $\mathrm{col}(\pi(j)) \neq \mathrm{col}_i$. We have that $\pi(j)$ is uniform on $\{1, \dots, \mathcal{T}\} \setminus \{j\}$. Since $\mathrm{col}(j) = \mathrm{col}_i$ there are $a_i - 1$ balls $k \in \{1, \dots, \mathcal{T}\} \setminus \{j\}$ for which $\mathrm{col}(k) = \mathrm{col}_i$. So,

$$\begin{aligned} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] &= \frac{1}{2} \frac{(\mathcal{T} - 1) - (a_i - 1)}{\mathcal{T} - 1} \\ &= \frac{1}{2} \frac{\mathcal{T} - a_i}{\mathcal{T} - 1} \ , \end{aligned}$$

which implies that

$$\begin{aligned} \sum_{j \in C_i} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] &= a_i \frac{1}{2} \frac{\mathcal{T} - a_i}{\mathcal{T} - 1} \\ &= \frac{1}{2} \frac{1}{\mathcal{T} - 1} (\mathcal{T} a_i - a_i^2) \ . \end{aligned}$$

It follows that

$$\begin{aligned} \sum_{i=1}^{\mathcal{T}-1} \sum_{j \in C_i} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] &= \frac{1}{2} \frac{1}{\mathcal{T} - 1} \sum_{i=1}^{\mathcal{T}-1} (\mathcal{T} a_i - a_i^2) \\ &= \frac{1}{2} \frac{1}{\mathcal{T} - 1} (\mathcal{T} \sum_{i=1}^{\mathcal{T}-1} a_i - \sum_{i=1}^{\mathcal{T}-1} a_i^2) \\ &= \frac{1}{2} \frac{1}{\mathcal{T} - 1} (\mathcal{T}(\mathcal{T} - a_0) - \sum_{i=1}^{\mathcal{T}-1} a_i^2) \ . \end{aligned}$$

All in all we now have that

$$\begin{aligned}
\sum_{i=0}^{\mathcal{T}-1}\sum_{j\in C_i} \mathrm{E}\left[\mathrm{price}_{\mathrm{col}}(j)\right] &= \frac{1}{2}a_0 + \frac{1}{2}\frac{1}{\mathcal{T}-1}(\mathcal{T}(\mathcal{T}-a_0) - \sum_{i=1}^{\mathcal{T}-1} a_i^2) \\
&= \frac{1}{2}a_0 - \frac{1}{2}\frac{\mathcal{T}}{\mathcal{T}-1}a_0 + \frac{1}{2}\frac{1}{\mathcal{T}-1}(\mathcal{T}\mathcal{T} - \sum_{i=1}^{\mathcal{T}-1} a_i^2) \\
&= \frac{1}{2}(\frac{-a_0}{\mathcal{T}-1}) + \frac{1}{2}\frac{1}{\mathcal{T}-1}(\mathcal{T}^2 - \sum_{i=1}^{\mathcal{T}-1} a_i^2) \\
&= \frac{1}{2}\frac{\mathcal{T}^2}{\mathcal{T}-1} - \frac{1}{2}\frac{1}{\mathcal{T}-1}(a_0 + \sum_{i=1}^{\mathcal{T}-1} a_i^2)\ .
\end{aligned}$$

To minimize this expression we have to maximize $a_0 + \sum_{i=1}^{\mathcal{T}-1} a_i^2$. Recall that col_0 is defined to be the most common color, so we must adhere to $a_0 \geq a_i$ for $i > 0$. Under this restriction it is easy to see that $a_0 + \sum_{i=1}^{\mathcal{T}-1} a_i^2$ is maximal when $a_0 = a_1 = \mathcal{T}/2$ and $a_2 = \cdots a_{\mathcal{T}} = 0$, in which case it has the value $\mathcal{T}/2 + (\mathcal{T}/2)^2$. So,

$$\begin{aligned}
\mathrm{E}\left[\mathrm{price}_{\mathrm{col}}\right] &= \frac{1}{2}\frac{\mathcal{T}^2}{\mathcal{T}-1} - \frac{1}{2}\frac{1}{\mathcal{T}-1}(\mathcal{T}/2 + (\mathcal{T}/2)^2) \\
&= \frac{1}{2}\frac{\mathcal{T}^2 - \mathcal{T}/2 + (\mathcal{T}/2)^2}{\mathcal{T}-1} \\
&= \frac{1}{2}\frac{4\tau^2 - \tau - \tau^2}{2\tau - 1} = \frac{1}{2}\frac{3\tau^2 - \tau}{2\tau - 1} = \frac{1}{2}\tau\frac{3\tau - 1}{2\tau - 1} > \frac{1}{2}\tau\frac{3\tau}{2\tau} = \frac{3}{4}\tau = \kappa\ .
\end{aligned}$$

E Efficient OT Extension

In this section 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 κ, the security parameter.

A similar result was proved in [HIKN08]. In [HIKN08] the amortized work is linear in κ too, but our constants are much better than those of [HIKN08]. In fact, our constants are small enough to make the protocol very practical.[7] Since [HIKN08] does not attempt to analyze the exact complexity of the result, it is hard to give a concrete comparison, but since the result in [HIKN08] goes over generic secure multiparty computation of non-trivial functionalities, the constants are expected to be huge compared to ours.

Let κ be the security parameter. We show that $\mathrm{OT}(\ell, \kappa)$ is linear reducible to $(\mathrm{OT}(\frac{8}{3}\kappa, \kappa), \mathrm{EQ}(\frac{4}{3}\kappa^2))$ for any $\ell = \mathrm{poly}(\kappa)$, i.e., given $\frac{8}{3}\kappa$ active-secure OTs of κ-bit strings we can produce an essentially unbounded number of active-secure OTs of κ-bit strings. The amortized work involved in each of these ℓ OTs is linear in κ, which is optimal.

The approach is as follows.

1. Use $\mathrm{OT}(\frac{8}{3}\kappa, \kappa)$ and a pseudo-random generator to implement $\mathrm{OT}(\frac{8}{3}\kappa, \ell)$.

[7] As an example, our test run (see Sect. 7) with $\ell = 54$ involved generating 44,826,624 aBits, each of which can be turned into one OT using two applications of a hash function. The generation took 85 seconds. Using these numbers, gives an estimate of 527,372 actively secure OTs per second. Note, however, that the generation involved many other things than generating the aBits, like combining them to aOTs and aANDs.

2. Use $\mathrm{OT}(\frac{8}{3}\kappa, \ell)$ and $\mathrm{EQ}(\frac{4}{3}\kappa\ell)$ to implement $\mathrm{WaBit}^{\mathcal{L}}$ for ℓ authentications with $\frac{4}{3}\kappa$-bit keys and MACs and with $\mathcal{L}$ being κ-secure.
3. Use a random oracle $H : \{0,1\}^{\frac{4}{3}\kappa} \to \{0,1\}^\kappa$ and $\mathrm{WaBit}^{\mathcal{L}}$ for ℓ authentications with $\frac{4}{3}\kappa$-bit keys to implement $\mathrm{OT}(\ell, \kappa)$, as described below.

Here, as in [HIKN08], we consider a hashing of $O(\kappa)$ bits to be linear work. The pseudo-random generator can be implemented with linear work using H.

From WaBit *to* OT. As a first step, we notice that the aBit box described in Sect. 4.2 resembles an intermediate step of the passive-secure OT extension protocol of [IKNP03]: an aBit can be seen as a random OT, where all the sender's messages are correlated, in the sense that the XOR of the messages in any OT is a constant (the global key of the aBit). This correlation can be easily broken using the random oracle. In fact, even if few bits of the global difference Δ leak to the adversary, the same reduction is still going to work (for an appropriate choice of the parameters). Therefore, we are able to start directly from the box for authenticated bits with weak key, or WaBit described in Sect. 4.1.

1. For the sender S the box samples $X_{i,0}, X_{i,1} \in_{\mathrm{R}} \{0,1\}^\kappa$ for $i = 1, \ldots, \ell$. If S is corrupted, then it gets to specify these inputs.
2. For the receiver R the box samples $\boldsymbol{b} = (b_1, \ldots, b_\ell) \in_{\mathrm{R}} \{0,1\}^\ell$. If R is corrupted, then it gets to specify these inputs.
3. The box outputs $((X_{1,b_1}, b_1), \ldots, (X_{\ell,b_\ell}, b_\ell))$ to R and $((X_{1,0}, X_{1,1}), \ldots, (X_{\ell,0}, X_{\ell,1}))$ to S.

Fig. 25. The Random OT box $\mathrm{ROT}(\ell, \kappa)$

1. Call $\mathrm{WaBit}^{\mathcal{L}}(\ell, \frac{4}{3}\kappa)$. The output to R is $((M_1, b_1), \ldots, (M_\ell, b_\ell))$. The output to S is $(\Delta, K_1, \ldots, K_\ell)$.
2. R computes $Y_i = H(M_i)$ and outputs $((Y_1, b_1), \ldots, (Y_\ell, b_\ell))$.
3. S computes $X_{i,0} = H(K_i)$ and $X_{i,1} = H(K_i \oplus \Delta)$ and outputs $((X_{1,0}, X_{1,1}), \ldots, (X_{\ell,0}, X_{\ell,1}))$.

Fig. 26. The protocol for reducing $\mathrm{ROT}(\ell, \kappa)$ to $\mathrm{WaBit}^{\mathcal{L}}(\ell, \frac{4}{3}\kappa)$

Here κ is the security level, i.e., we want to implement OT with insecurity $\mathrm{poly}(\kappa)2^{-\kappa}$. We are to use an instance of $\mathrm{WaBit}^{\mathcal{L}}$ with slightly larger keys. Specifically, let $\tau = \frac{4}{3}\kappa$, as we know how to implement a box $\mathrm{WaBit}^{\mathcal{L}}$ with τ-bit keys and where $\mathcal{L}$ is κ-secure for $\kappa = \frac{3}{4}\tau$. We implemented such a box in Sect. 4.1. The protocol is given in Fig. 26. It implements the box for random OT given in Fig. 25.

We have that $M_i = K_i \oplus b_i\Delta$, so $Y_i = H(M_i) = H(K_i \oplus b_i\Delta) = X_{i,b_i}$. Clearly the protocol leaks no information on the b_i as there is no communication from R to S. It is therefore sufficient to look at the case where R is corrupted. We are not going to give a simulation argument but just show that $X_{i,1\oplus b_i}$ is uniformly random to R except with probability $\mathrm{poly}(\kappa)2^{-\kappa}$.

Since $X_{i,1\oplus b_i} = H(K_i \oplus (1 \oplus b_i)\Delta)$ and H is a random oracle, it is clear that $X_{i,1\oplus b_i}$ is uniformly random to R until R queries H on $Q = K_i \oplus (1 \oplus b_i)\Delta$. Since $M_i = K_i \oplus b_i\Delta$ we have that $Q = K_i \oplus (1 \oplus b_i)\Delta$ would imply that $M_i \oplus Q = \Delta$. So, if we let R query H, say, on $Q \oplus M_i$ 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 R until it queries H on Δ. It is not hard to show that the probability with which an adversary running in time $t = \mathrm{poly}(\kappa)$ can ensure that $\mathrm{WaBit}^{\mathcal{L}}$ does not fail and then query H on Δ is $\mathrm{poly}(\kappa)2^{-\kappa}$. This follows from the κ-security of $\mathcal{L}$.

F Proof of Thm. 5

The simulator answers global key queries to the dealer by doing the identical global key queries on the ideal functionality LaOT(ℓ) and returning the reply from LaOT(ℓ). This gives a perfect simulation of these queries, and we ignore them below.

For honest sender and receiver correctness of the protocol follows immediately from correctness of the aBit box and the EQ box.

Lemma 13. *The protocol in Fig. 15 securely implements* LaOT(ℓ) *against corrupted* A.

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

1. First Sim 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 Sim samples a bit $y \in_{\mathrm{R}} \{0,1\}$, 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 a LaOT box. The box 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 $j \in \{0,1\}$ Sim tests if $(\overline{M}_{x_j}, \overline{x}_j) = (M_{x_j}, x_j)$. If, for some j, this is not the case Sim inputs a guess to the LaOT box guessing that $c = (1-j)$ to the LaOT box. If the box outputs `fail` Sim does the same and aborts the protocol. Otherwise Sim proceeds by sending y to A^*. Notice that if Sim 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 Sim 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 Sim receives A^*'s input (T_0, T_1) for the EQ box 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 Sim inputs a guess to the LaOT box guessing that $c = (1-j)$. If the box outputs `fail`, Sim 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 $M^*_{x_j} \in \{0,1\}^\kappa$ 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 LaOT box 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 Sim 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 $r \in_{\mathrm{R}} \{0,1\}$ 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 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 14. *The protocol in Fig. 15 securely implements* $\mathrm{LaOT}(\ell)$ *against corrupted* B.

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

1. The simulation starts by Sim getting B^*'s input to dealer Δ_A, (M_c, c), (M_r, r), K_{x_0} and K_{x_1}. Then Sim simply inputs Δ_A, (M_c, c), $M_z = M_r$, K_{x_0} and K_{x_1} to the LaOT box. The box outputs z to Sim 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 Sim samples random keys $T_0, T_1 \in_{\mathrm{R}} \{0,1\}^{\kappa}$. Since Sim 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 $X_{1\oplus c} \in_{\mathrm{R}} \{0,1\}^{2\kappa+1}$ and inputs (X_0, X_1) to B^*.
3. The corrupt receiver B^* replies by sending some $\overline{y} \in \{0,1\}$.
4. Sim sets $\overline{z} = r \oplus \overline{y}$, computes $I_{\overline{z}} = H(M_z) \oplus T_{1\oplus\overline{z}}$ and samples $I_{1\oplus\overline{z}} \in_{\mathrm{R}} \{0,1\}^{\kappa}$. It then inputs (I_0, I_1) to B^*.
5. B^* outputs some $(\overline{T}_0, \overline{T}_1)$ for the EQ box and Sim 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 $Q = (Q_1, \dots, Q_q)$ of all B^*'s queries to H. Since B^* is efficient we have that q is a polynomial in κ. 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 Thm. 7 in App. I using the programability of the RO. □

G Proof of Thm. 6

We want to show that the protocol in Fig. 16 produces secure aOTs, having access to a box that produces leaky aOTs. Remember that a leaky aOT or LaOT, is insecure in the sense that a corrupted sender can make guesses at any of the choice bits: if the guess is correct, the box does nothing and therefore the adversary knows that the guess was correct. If the guess is wrong, the box alerts the honest receiver about the cheating attempt and aborts.

In the protocol the receiver randomly partitions ℓB leaky OTs in ℓ buckets of size B. First we want to argue that the probability that every bucket contains at least one OT where the choice bit is unknown to the adversary is overwhelming. Repeating the same calculations as in the proof of Thm. 8 it turns out that this happens with probability bigger than $1 - (2\ell)^{(1-B)}$.

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 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 Thm. 8 in App. H.

H Proof of Thm. 7

Proof. The simulator answers global key queries to the dealer by doing the identical global key queries on the ideal functionality LaAND(ℓ) and returning the reply from LaAND(ℓ). 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 aBit box.

Lemma 15. *The protocol in Fig. 19 securely implements the* LaAND *box 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 Sim for honest B.

1. First Sim receives A^*'s input $(M_x, x), (M_y, y), (M_r, r)$ for the dealer.
   Then Sim receives the bit $d \in_{\mathrm{R}} \{0,1\}$.
2. Sim samples a random $U \in_R \{0,1\}^{2\kappa}$ and sends it to A^*. Then Sim reads $\overline{V}$, A^*'s input to the EQ box. If $\overline{V} \neq (1-x)H(M_x, M_z) \oplus x(U \oplus H(M_x, M_y \oplus M_z))$ or $d \oplus y \neq xy$, Sim outputs abort, otherwise, it inputs $(x, y, z, M_x, M_y, M_z = M_r)$ to the LaAND box.

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 is 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 the oracle on $(K_x, K_z) = (M_x, M_z \oplus \Delta_A)$, and since Sim knows the outputs of corrupted A, which include M_z, and see the input $M_z \oplus \Delta_A$ to the RO H, if A^* queries the oracle on $(K_x, K_z) = (M_x, M_z \oplus \Delta_A)$, Sim 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 Sim 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 Sim 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 $(Q_1, Q_2) \in \{0,1\}^{2\kappa}$, 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 aBit$(3\ell, \kappa)$ to determine if $\Delta = \Delta_A$. If Sim learns Δ_A this way, she proceeds as described now. Note that since A is corrupted, Sim knows all outputs to A, i.e., Sim knows all MACs M and all bits b. If $b = 0$, then Sim also knows the key, as $K = M$ when $b = 0$. If $b = 1$, Sim computes the key as $K = M \oplus \Delta_A$. So, when Sim 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 Sim 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 Sim 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, Sim simply set $H(K_x, K_z) \leftarrow U \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z)$. □

Lemma 16. *The protocol described in Fig. 19 securely implements the* LaAND *box against corrupted* B.

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

1. The simulation starts by Sim 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. Sim receives $\overline{U}$ from B^*, and reads $\overline{V}$, B^*'s input to the equality box.
4. If $\overline{U} = H(K_x, K_z) \oplus H(K_x \oplus \Delta_A, K_y \oplus K_z)$ and $\overline{V} = H(K_x, K_z)$, input (K_x, K_y, K_z) to the box for LaAND and complete the protocol (this is the case where B^* is behaving as an honest player). 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 $\overline{V} = \overline{U} \oplus H(K_x \oplus \Delta_A, K_z \oplus K_z)$, input $g = 0$ or $g = 1$ resp. into the LaAND box as a guess for the bit x. If the box 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, that is uniformly distributed both in the real game and in the simulation, and in the aborting condition, that is the same in the real and in the simulated game. □

I Proof of Thm. 8

Proof. The simulator answers global key queries to LaAND($B\ell$) by doing the identical global key queries on the ideal functionality aAND(ℓ) 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 LaAND box, 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 LaAND($B\ell$).

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 LaAND(ℓ'). 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 $2 \le B < \ell$ we have that

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

so

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

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 LaAND box. 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 aAND box. 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 $d \in_{\mathrm{R}} \{0,1\}$, 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 aAND box on behalf of B^*. Finally the aAND box 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 LaAND box 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 aAND box exactly as they would be in the real world the simulation will be perfect.

$\square$

J Full Overview Diagram

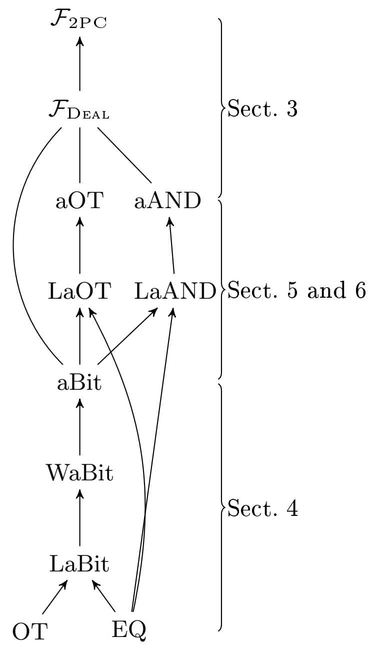

Fig. 27. Full paper outline.