Leakage-Tolerant Interactive Protocols[⋆]

Nir Bitansky[1,2], Ran Canetti[1,2], and Shai Halevi[3]

[1] Tel Aviv University
[2] Boston University
[3] IBM T.J. Watson Research Center

Abstract. We put forth a framework for expressing security requirements from interactive protocols in the presence of arbitrary leakage. This allows capturing different levels of leakage-tolerance of protocols, namely the preservation (or degradation) of security, under coordinated attacks that include various forms of leakage from the secret states of participating components. The framework extends the universally composable (UC) security framework. We also prove a variant of the UC theorem that enables modular design and analysis of protocols even in face of general, non-modular leakage.
We then construct leakage-tolerant protocols for basic tasks, such as secure message transmission, message authentication, commitment, oblivious transfer and zero-knowledge. A central component in several of our constructions is the observation that resilience to adaptive party corruptions (in some strong sense) implies leakage-tolerance in an essentially optimal way.

1 Introduction

Traditionally, cryptographic protocols are studied in a model where participants have a secret state that is assumed to be completely inaccessible by the adversary. In this model, the adversary can only influence the system via anticipated interfaces (such as, the communication among parties). These interfaces are crossed only when the adversary manages to fully corrupt a party, thus gaining access to its entire inner state.

In reality, an intermediate setting often emerges, when the adversary manages to gain some partial information on the secret state of uncorrupted parties. This information, termed *leakage*, can be obtained by a variety of side channels attacks that bypass the usual interfaces and are often undetectable. Known examples include: timing, power, EM-emission, and cache attacks (see [Sta09] for a survey).

The threat of leakage gained much attention in the past few years, giving rise to an impressive array of *leakage-resilient* schemes for basic cryptographic tasks such as encryption and signatures, as well as general non-interactive circuits (e.g., [DP08, AGV09, ADW09, DKL09, Pie09, NS09, ADN$^+$10, BKKV10,

[⋆] Supported by the Check Point Institute for Information Security. The first two authors are also supported by Marie Curie grant PIRG03-GA-2008-230640, and ISF grant 0603805843.

DHLAW10b, DHLAW10a, BSW11]). Most of the work concentrates on preserving, in the presence of leakage, the same functionality and security guarantees that the original primitives guarantee in a leak-free setting. Such strong leakage-resilience is typically guaranteed only when the leakage is restricted in some ways. Examples include: assuming bounded amounts of leakage, assuming that leakage only occurs in specific times (e.g., prior to encryption), or assuming that leakage is limited to specific parts of the state, such as the active parts in the *only computation leaks* model [MR04].

However, in many cases maintaining the same level of security as in a leak-free setting may be too costly, or even outright impossible. To exemplify this, consider the task of *secure message transmission* (SMT), where a sender wishes to transmit a (secret) message m to a receiver, so that the contents of m remain completely hidden from any adversary witnessing the communication. In the leak-free setting, the problem is easily solved using standard semantically secure encryption; however, in the presence of leakage, this is no longer the case. In fact, semantic security is not achievable at all: an adversary that can get even one bit of arbitrary leakage, from either party, can certainly learn any bit of the message, since this bit must reside in the party's leaky memory at some point.

Nevertheless, this inherent difficulty does not imply that we should give up on security altogether, but rather that we should somehow meaningfully relax the security requirements from protocols in the presence of leakage. Concretely, in the above example, we would like to design schemes in which one-bit of leakage on the message does not compromise the security of the entire message. More generally, we would like to establish a framework that will allow to express and analyze security of general cryptographic tasks in the presence of general (non-restricted) leakage, where the level of security may gracefully degrade according to the amount of leakage (that might develop over time). A first step in this direction was taken by Halevi and Lin [HL11] in the context of encryption.

Another intriguing question is what are the composability properties of resilience to leakage. Can one combine two or more schemes and deduce leakage-resilience of the combined system based only on the leakage-resilience properties of the individual schemes? If so, constructs with various levels of leakage-resilience may be composed to obtain new systems that enjoy improved such resilience properties. Some specific examples where this is the case have been recently exhibited [BCG$^+$11, BGK11, GJS11]. What can we say in general?

1.1 Our Contribution

We propose a new approach for defining leakage-resilience, or rather *leakage-tolerance*, properties of cryptographic protocols. The approach is based on the ideal model paradigm and, specifically, on the UC framework. The approach allows formulating relaxed security properties of protocols in face of leakage and, in particular, allows specifying how the security of protocols degrades with leakage. It also allows specifying leakage-tolerant variants of interactive, multi-party protocols for general cryptographic tasks. In this context, the new modeling also captures attacks that combine leakage with other "network based" attacks such as controlling the communication and corrupting parties. In addition:

- We prove a general security-preserving composition theorem with respect to the proposed notion. This allows constructing and analyzing protocols in a modular way *while preserving leakage-tolerance properties.* This is a powerful tool, given the inherently modularity-breaking nature of leakage attacks.
- We describe a methodology for constructing leakage-tolerant protocols in this framework. Essentially, we show that any protocol that is secure against adaptive party corruptions (in some strong sense) is already leakage-tolerant.
- Using the above methodology and other techniques, we construct composable leakage-tolerant protocols for secure channels, commitment, zero-knowledge, and honest-but-curious oblivious transfer. (commitment and zero-knowledge are realized in the common reference string model.)

Below we describe these contributions in more detail.

Leakage-tolerant security within the ideal model paradigm. Following the ideal model paradigm, we define security by requiring that the protocol π at hand provides the same security properties as in an "ideal world" where processing is done by a trusted party running some functionality $\mathcal{F}$. Specifically, in the UC framework, a protocol π UC-realizes a functionality $\mathcal{F}$ if for any adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can tell whether it is interacting with $\mathcal{A}$ and π or with $\mathcal{S}$ and $\mathcal{F}$.

We consider a "real world" where the adversary can get leakage on the state of any party at any time. As we argued above, such attacks may unavoidably degrade the security properties of the protocols at hand and to account for this degradation we also allow leakage from the trusted party in the ideal world. Specifically, the functionality $\mathcal{F}$ defines the "ideal local state" for each party and the party's behavior (and degradation in security) after leakage. (Typically, we will be interested in functionalities where the ideal local state includes the party's inputs and outputs, but weaker functionalities that allow joint leakage on the inputs of several parties can also be considered.) When $\mathcal{A}$ performs a leakage measurement L on the state of some party in the real protocol π, the simulator $\mathcal{S}$ is entitled to a leakage measurement L' on the ideal local state of that party in the ideal protocol. We allow the simulator to choose any function L', so long that its output length is the same as that of L.

For example, we allow our leaky SMT functionality to leak bits from message that it sends and require that a real world attacker that gets ℓ bits of leakage from the state of the implementation can be simulated by a simulator that learns only ℓ bits about the message. Our model also allows the functionality to react to leakage, in order to handle situations where security is only maintained as long as not too much leakage occurred. (For example, an authenticated channels functionality may allow forgeries once the attacker gets more bits of leakage than the security parameter, but not before that.)

Leakage vs. adaptive corruptions for secure channels. Consider trying to realize leaky SMT in our model using standard encryption; namely, the receiver sends its public key to the sender, who sends back an encryption of m. In the ideal world, the simulator does not witness any communication (and has no information about the message), so it can simulate the cipher by encrypting say the all-zero string, which should be indistinguishable from an encryption of m. However, after seeing the ciphertext the adversary $\mathcal{A}$ can ask for a leakage query specifying (say) the entire secret decryption key and the first bit of m. Although the simulator can now ask for many bits of leakage on m, it can no longer modify the ciphertext that it sent before and therefore cannot maintain a consistent simulation.

A similar problem arises in the well studied setting of *adaptive corruption* (with non-erasing parties), where the adversary can adaptively corrupt parties throughout the protocol and learn their entire state. Also there, the simulator needs to first generate some messages (e.g., the ciphertext) without knowing the inputs of the parties (e.g., the message m), and later it learns the inputs and has to come up with an internal state that explains the previously-generated messages in terms of these inputs. Indeed, it turns out that techniques for handling adaptive corruptions can be used to get leakage-tolerance.

In fact, the problem of secure leaky channels can be solved simply by plugging in *non-committing encryption* (NCE) [CFGN96], which was developed for adaptively secure communication. Recall that an NCE scheme allows generating a "fake" equivocal ciphertext $\tilde{c}$ that can later be "opened" as an encryption of any string of a predefined length ℓ. Namely, $\tilde{c}$ is generated together with a poly-size equivocation circuit E, such that, given any message $m \in \{0,1\}^\ell$, $E(m)$ generates randomness $(\tilde{r}_S^m, \tilde{r}_R^m)$, for both the sender and the receiver, that "explains" $\tilde{c}$ as an encryption of m.

To obtain leakage-tolerant secure message transmission, we can simply encrypt the message using an NCE scheme. The simulator can now generate the fake ciphertext $\tilde{c}$ with the associated equivocation circuit E and can then translate any c-dependent leakage function on the entire state (plaintext and randomness) into a leakage function on the plaintext only, which can be queried to the leaky SMT functionality. When leakage on $P \in \{S, R\}$ occurs, the simulator $\mathcal{S}$ translates the leakage function $L(m, r_P)$ into $L'(m) = L(m, E(m)) = L(m, \tilde{r}_P^m)$. Indeed, this idea was used in [BCG$^+$11] in the context of a specific protocol.

The general case. The above example can be made general. Specifically, we show that, with some limitations, any protocol that realizes a functionality $\mathcal{F}$ under adaptive corruptions also realizes a leaky variant $\mathcal{F}^{+\mathsf{lk}}$ under leakage. The "leaky variant" is a natural adaptation that allows leakage on the state of $\mathcal{F}$, just like the leaky SMT allow leakage on the transmitted message. This variant, denoted $\mathcal{F}^{+\mathsf{lk}}$, is identical to $\mathcal{F}$ except that $\mathcal{F}^{+\mathsf{lk}}$ allows the simulator to apply arbitrary leakage functions to the ideal local state (which is the same as the state defined in a semi-honest corruption). When such leakage occurs the environment is reported on the identity of the leaking party and the number of bits leaked. (This makes sure that the simulator can only leak the same number of bits as in the protocol execution.) After such a leakage event, $\mathcal{F}^{+\mathsf{lk}}$ behaves in the same way that $\mathcal{F}$ behaves after a semi-honest corruption of that party. That is, if $\mathcal{F}$ modifies its overall behavior following the corruption of a party, then $\mathcal{F}^{+\mathsf{lk}}$ modifies its behavior in the same way. (In the applications considered in this work, we will consider functionalities that do not change their behavior after semi-honest corruptions, see Section 5.)

A limitation of this result is that it only holds when the given proof of security uses a restricted type of simulators, namely ones that work "obliviously" of the state that they learn when corrupting a player. We call such simulators *corruption-oblivious.* We have:

Theorem 1.1 (informal). *If protocol π realizes $\mathcal{F}$ under adaptive corruptions (either semi-honest or Byzantine) with a corruption-oblivious simulator, then it also realizes $\mathcal{F}^{+\mathsf{lk}}$ under arbitrary leakage (and the same type of corruptions).*

Composable leakage-tolerance. An important property of ideal model based notions of security is that they enable modularity, since the guarantees that they provide are preserved even under (universal) composition of protocols. That is, if a protocol π realizes an ideal functionality $\mathcal{F}$, the security properties of $\mathcal{F}$ carry over to any environment where π is used.

To achieve such modularity, common models of composable security rely crucially on viewing different sub-modules of a large system as autonomous small systems, each with its own local state and well-defined interfaces to the rest of the system. Unfortunately, extending this "modular security" paradigm to the leaky world is problematic: real world leakage is inherently non-modular, in that the adversary can obtain leakage from the joint state of an entire physical device and is not bound by our modular separation to logical modules of the software running on the device. In fact, it is not even clear how to *express* joint leakage from the state of different modules within standard models, let alone how to argue about preservation of security properties.

We extend the UC security framework [Can01] to allow expressing leakage attacks from physical devices that span multiple logical modules. We first allow the protocol analyzer to delineate sets of "jointly leakable modules" (roughly corresponding to physical machines). Then, we introduce a new entity, called an aggregator, that has access to the internal states of all the modules in each set.

To get leakage from the joint state of the modules in a set P, the adversary sends the leakage function L to the aggregator, who applies L to the combined state and returns the result to the adversary. The same mechanism is used to obtain leakage from ideal functionalities, except that here the ideal functionality $\mathcal{F}$ hands the aggregator some "ideal local state" that $\mathcal{F}$ associated with the set P. We stress again that our model considers a strong adversary that obtains leakage information in a non-modular way from multiple subroutines that reside on a common device, this makes positive results in this model quite strong.

Having extended the model of protocol execution to capture leakage attacks, we would like to re-assert the composability property described above, i.e., to re-prove the UC composition theorem from [Can01] in our setting. However, that theorem was only proved for systems that behave in a "modular way", and the proof no longer holds in the presence of our modularity-breaking aggregator.

Still, we manage to salvage much of the spirit of the UC theorem, as follows. We formulate a more stringent variant of UC security by putting some technical restrictions on the simulator and then re-assert the UC theorem with respect to this varaint. Similarly to the case of corruption-oblivious simulators, here too we require that the simulator $\mathcal{S}$ handles leakage queries "obliviously".

Roughly, $\mathcal{S}$ has a "query-independent" way of translating, via a state-translation function, real world leakage queries $L(\mathsf{state}_\pi)$ to ideal world leakage queries $L'(\mathsf{state}_\mathcal{F})$. Furthermore, it ignores the leakage-results in the rest of the simulation. We call such simulators *leakage-oblivious* and show:

Theorem 1.2 (UC-composition with leakage, informal). *Let $\rho^\mathcal{F}$ be a protocol that invokes $\mathcal{F}$ as a sub-routine. Let π be a protocol that UC-emulates $\mathcal{F}$ with a leakage-oblivious simulator. Then the composed protocol $\rho^{\pi/\mathcal{F}}$ (where each call to $\mathcal{F}$ is replaced with a call for π) UC-emulates $\rho^\mathcal{F}$ in face of leakage. Furthermore, it does so with a leakage-oblivious simulator.*

Theorem 1.2 provides a powerful tool in the design of leakage-resilient protocols. In particular, we later use it to (a) combine any leakage-resilient protocol that assumes authenticated communication with a leakage-resilient authentication protocol into a leakage-resilient protocol over unauthenticated channels, and (b) to combine any leakage-resilient zero-knowledge protocol that assumes ideal commitment with leakage-resilient commitment protocols to obtain a composite leakage-resilient zero-knowledge protocol.

Leakage-tolerant protocols. We construct leakage-tolerant protocols for a number of basic cryptographic tasks. We first observe that the general result regarding the leakage-tolerance of adaptively secure protocols (Theorem 1.1) in fact guarantees UC security *with leakage-oblivious simulators*. We then observe that existing adaptively secure protocols for secure channels, UC commitment and UC semi-honest oblivious transfer already have corruption-oblivious simulators; hence, we immediately get:

- Assume authenticated communication. Then, any non-committing encryption scheme UC-realizes $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{SMT}}$ in the presence of arbitrary leakage using a leakage-oblivious simulator.
- In the CRS model, the UC commitment protocols of Canetti and Fischlin [CF01] and Canetti, Lindell, Ostrovsky and Sahai [CLOS02], UC-realize $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{MCOM}}$ (the leaky version of the multi-instance commitment functionality) in the presence of arbitrary leakage. Furthermore, they do so with leakage-oblivious simulators.
- Also in the CRS model, the UC (non-interactive) zero-knowledge protocol of Groth, Ostrovsky and Sahai[GOS06] realize $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{ZK}}$ under arbitrary leakage.
- The semi-honest oblivious transfer protocol of [CLOS02] for adaptive corruptions UC-realizes $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{OT}}$ (the leaky version of the ideal oblivious transfer functionality in the presence of arbitrary leakage). Furthermore, it does so with leakage-oblivious simulators.

In this work, we do not consider the generation of a CRS in the presence of leakage; rather, we treat the CRS as an external entity that can be generated in a physically separate location. As in other settings, here too it is interesting to find ways to reduce the setup requirements.

Finally, we note that for certain functionalities $\mathcal{F}$, applying Theorem 1.1 alone may still not give an adequate level of leakage-resilience. Indeed, while the leaky adaptation $\mathcal{F}^{+\mathsf{lk}}$ assures graceful degradation of privacy, it may not account for correctness (or soundness) aspects in the face of leakage. In such cases, we may need to further strengthen $\mathcal{F}^{+\mathsf{lk}}$. One such example is message authentication. Indeed, $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{AUTH}}$ gives essentially no security guarantees: as soon as even a single bit of information is leaked from the sender, $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{AUTH}}$ behaves as if the sender is fully corrupted, in which case forgery of messages is allowed. We thus first formulate a variant of $\mathcal{F}_{\mathsf{AUTH}}$ that guarantees authenticity as long as the number of bits leaked is less than some threshold B. We then realize this functionality, denoted $\mathcal{F}^{+B}_{\mathsf{AUTH}}$, assuming an initial k-bit shared secret key between the parties and as long as at most $B = O(k)$ bits leak *between each two consecutive message transmissions.* Furthermore, we do this with a leakage-oblivious simulator. The techniques used to realize $\mathcal{F}^{+B}_{\mathsf{AUTH}}$ include information-theoretic leakage-resilient message authentication codes, as well as NCE schemes.

We note that the techniques here borrow strongly from the techniques used in [BCG$^+$11] for the related goal of authentication within the context of obfuscation with leaky hardware. That work, however, analyzed these tools in an ad-hoc manner, and the results there apply only to that specific context.

In contrast, using the above UC theorem with leakage, we can combine the above authentication protocol with any protocol that assumes ideally authenticated communication to obtain composite leakage-tolerant protocols that withstand unauthenticated communication.

Finally, we address the task of obtaining zero-knowledge from ideal leaky commitment $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{MCOM}}$ (the adaptive NIZK protocol of [GOS06] is obtained from specific number-theoretic assumptions on bilinear groups). At first it may seem that, as in the case of commitment, existing protocols for UC-realizing the ideal zero-knowledge functionality, $\mathcal{F}_{\mathsf{ZK}}$, would work also in the case of leakage. However, this turns out not to be the case. In particular, while the protocol of [CF01] for UC-realizing $\mathcal{F}_{\mathsf{ZK}:R}$, for some relation R, given $\mathcal{F}_{\mathsf{MCOM}}$ is indeed secure against adaptive corruptions, the simulator turns out not to be corruption-oblivious and Theorem 1.1 does not apply.

Instead, we settle for UC-realizing, in the presence of leakage, a weaker variant of $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{ZK}:R}$. This weaker variant permits violation of the soundness requirements if too many bits were leaked from the verifier. We denote this weaker version by $\mathcal{F}^{+B}_{\mathsf{ZK}:R}$, where B is the leakage threshold for the verifier. We show how to UC-realize $\mathcal{F}^{+B}_{\mathsf{ZK}:R}$ for $B = k - \omega(\log k)$ (where k is the security parameter), given access to $\mathcal{F}^{+\mathsf{lk}}_{\mathsf{MCOM}}$. Using the (leaky) universal composition theorem and the protocol for realizing $\mathcal{F}_{\mathsf{MCOM}}$ (mentioned above), we obtain a protocol for UC-realizing $\mathcal{F}^{+B}_{\mathsf{ZK}:R}$ in the CRS model.

Concurrent work. Garg, Jain, and Sahai [GJS11] also investigate zero-knowledge in the presence of leakage, albeit not in the UC setting. Instead, they consider a stand-alone definition with a rewinding simulator (where a CRS is not needed). Some of the difficulties that emerge in standard 3-round zero-knowledge protocols, as well as the suggestion to overcome them using the Goldriech-Kahn paradigm, were communicated to us by Amit Sahai.

Damgård, Hazay, and Arpita [DHP11] consider leakage-resilient two-party protocols. Their definition of security, which is also ideal-model based, accounts also for noisy leakage (namely leakage that might not be length-restricted, but is somewhat entropy preserving). They achieve leakage-resilience (or tolerance) for NC_1 functions in a setting where one party is statically and passively corrupted and the other party is leaky. The result, however, only applies in the "only computation leaks" (OCL) model of [MR04] (and with some extra technical limitations). They also prove a security preserving composition theorem, but their modeling considers only separate leakage from each module (rather than overall leakage as considered here). They also construct a leakage-tolerant OT protocol for sufficiently entropic inputs distributions, but only in the OCL model and under a relatively strong hardness assumption; in terms of communication, however, their protocol is more efficient than ours. Finally, we remark that the setting where one party is statically passively corrupted can be seen as a special case of a weak leakage-tolerance model, where the ideal world simulator is allowed to jointly leak from all the parties. See further discussion in Section 5.3.

2 Modeling Leakage in the UC Framework

This section defines the new model of UC security with leakage. Here we provide a high-level overview, the full details can be found in the full version of this work [BCH11]. Recall that the basic UC framework considers realization of an "ideal specification" $\mathcal{F}$ by a "real implementation" π. (Formally both $\mathcal{F}$ and π are just protocols, we call them by different names to guide the intuition.) The realization requirement is that for any "real world attacker" $\mathcal{A}$ against the implementation π there exists another adversary $\mathcal{S}$ (called a simulator) against the specification $\mathcal{F}$, such that an "environment" $\mathcal{Z}$ that interacts with $\mathcal{S}, \mathcal{F}$ has essentially the same view as in an interaction with $\mathcal{A}, \pi$.

The basic UC execution model lets the environment $\mathcal{Z}$ determine the inputs to the parties running the protocol and see the outputs generated by these parties and also allows free communication between the environment and the adversary. The adversary, typically, has full control over the communication between parties and the ability to "corrupt" parties in various ways. Corruption is modeled as just another interface available to the adversary, where it can send a message "you are corrupted" to any party. (In the case of standard passive corruption, the party responds to this message by handing its entire internal state to the adversary. To model Byzantine corruption, the party also changes the program that it is running from then on.)

A crucial aspect of the UC framework is its modularity, where programs can call subroutines, and these subroutines are treated as separate entities that can be analyzed separately for security properties. Importantly, local randomness and secrets that are used by a subroutine should typically not be visible to the calling routine or to other components in the system.

A useful technicality in the UC framework, is that it is sufficient to prove security only with respect to the dummy real world adversary $\mathcal{D}$. This is the adversary that simply reports all the information it receives to the environment and follows all the instructions of the environment regarding sending messages to parties and ideal functionalities. Relying on the fact that any adversary can be emulated by the environment itself, it is easy to show that simulation of the dummy adversary $\mathcal{D}$ implies simulation for any adversary.

Leaky UC. A natural approach to modeling leakage within the UC framework is to view it as a weak form of corruption, where the adversary gets some information about the internal state of the leaky party but perhaps not all of it. Also, leakage resembles "semi-honest" corruption more than "malicious", in that leaky parties keep following the same protocol and do not change their behavior following a leakage event. Thus we could provide yet another interface to the adversary where it can send a "leak L" message to a party (where L is some function) and have that party reply with $L(s)$ where s is its internal state.

The leakage aggregator. A serious shortcoming of the modeling approach in the previous paragraphs is that it only lets the adversary obtain leakage on individual processes (or subroutines). In contrast, real life leakage usually provides information that depends on the entire state of a physical device, including all the processes that are currently running on it. To account for this inherently non-modular property of real life leakage, we introduce to the model a new "global entity" that we call the leakage aggregator. The aggregator $\mathcal{G}$ can access the entire internal state of all the components in the system. A leakage query specifies a leakage function L and a set of processes $P = \{p_1, \ldots, p_t\}$. This query is forwarded to the aggregator, who evaluates $L(s_1, \ldots, s_t)$ and returns the result to the adversary. Some important technicalities regarding the working of $\mathcal{G}$ are the following:

- A convention should be set for how to specify the sets of processes and ensure that this is a "legitimate set" for joint leakage. We assume that processes are tagged with "party identifiers" pid (roughly corresponding to physical machines), and joint leakage is allowed only from a set of processes that all have the same pid.
- As done for corruptions, here too the identity of the leaky processes and the amount of leakage needs to be reported to the environment. This forces the simulator, in the ideal world, to use the same amount of leakage from the same processes as in the real world.
- Since ideal functionalities represent idealized constructs that do not necessarily run on physical devices, they are often associated with more than one pid. Thus care should be taken when deciding how an ideal functionality reacts

to leakage queries w.r.t. one of its pid's. (For example, the secure-channels functionality runs on behalf of both the sender and the receiver, and would typically react differently to sender-leakage than to receiver-leakage queries.) We let the ideal functionality itself decide how to reply when $\mathcal{G}$ asks it for the state corresponding to any of its pid's. (This is the same convention as used for corruption, where the functionality gets to decide what to reveal to the adversary when one of its pid's is corrupted.) Typically, the "state" associated with a certain pid will be just the inputs that were received from that pid and the outputs it receives.

– To allow functionalities to react to leakage situations, we have $\mathcal{G}$, upon accessing the state of a module, report to that module the output size of the leakage function L. Typically, "real world implementations" ignore this report (since we assume that real world leakage is undetectable), but "ideal functionalities" may use it to change their behavior (e.g., reduce the security guarantee if too much leakage occurred).

With these conventions in place, a leakage operation is handled as follows: first, the adversary sends a query $(\mathsf{leak}, L, \mathsf{pid})$ to $\mathcal{G}$, where L is the leakage function and pid is the leaking party ID. Then, $\mathcal{G}$ obtains $\mathsf{state}_{\mathsf{pid}}$, the total state of party pid, applies L to $\mathsf{state}_{\mathsf{pid}}$, and returns the result to $\mathcal{A}$. Finally, $\mathcal{G}$ reports the output length of the function L to all the processes whose state is included in $\mathsf{state}_{\mathsf{pid}}$ and reports pid and the output length to the environment

We note that the security guarantee provided by this model may be weaker than one could desire, as the number of leaked bits is reported to *each one* of the processes (or functionalities). This means that when a domain leaks ℓ bits, each one of its components behaves as if the ℓ bits leaked entirely from this component. While this is a relatively weak leakage-resilience guarantee, it seems unavoidable in any general model with modularity-breaking leakage.

Leakage-oblivious simulation. Following the approach of basic UC security, the definition of protocol emulation requires that for any adversary $\mathcal{A}$ that attacks the implementation π there exists a simulator $\mathcal{S}$ that attacks the specification $\mathcal{F}$ so that no environment can distinguish between an interaction with $\mathcal{A}$ and π, and an interaction with $\mathcal{S}$ and $\mathcal{F}$. In particular, $\mathcal{S}$ must provide an overall transformation from one interaction scenario to the other, including among others the leakage queries made by $\mathcal{A}$ to the parties (via the aggregator). As noted above, an equivalent requirement considers, instead of any adversary $\mathcal{A}$, only the dummy adversary, $\mathcal{D}$, that merely passes messages between the environment and the protocol's parties.

This natural requirement, however, has (seemingly inherent) difficulties when considering composition of protocols. In particular, we were not able to prove a general composition theorem in this model (see details in Section 3). Consequently, we consider a more restricted notion of protocol emulation, which we term emulation with leakage-oblivious simulators.

To simplify the exposition, we describe here leakage-oblivious simulation only with respect to the dummy adversary $\mathcal{D}$. A leakage-oblivious simulator $\mathcal{S}$ for the dummy adversary has a special form: specifically, $\mathcal{S}$ has a separate subroutine $\tilde{S}$ for handling leakage. When $\mathcal{S}$ receives from the environment a request to apply a leakage function L to a set P of processes, $\tilde{S}$ is invoked to produce a "state translation" function T. This function is meant to transform the internal state of P in the specification $\mathcal{F}$ into "the actual state" in the implementation π. Once T is produced, the aggregator is given the composed leakage function $L \circ T$. Finally, when the leakage-result is returned, it is forwarded directly to the environment and $\mathcal{S}$ returns to its state prior to the leakage event.

The subroutine $\tilde{S}$ should operate **independently of the leakage function** L, its only input is the state of $\mathcal{S}$ (prior to the leakage query) and a party identifier pid. Also, the leakage operation has **no side effects on** $\mathcal{S}$. That is, following the leakage event $\mathcal{S}$ return to the state that it had before that event.

3 Universal Composition of Leaky Protocols

We now state the universal composition theorem for leaky protocols and leakage-oblivious simulators (as defined in Section 2). Let π be an implementation and $\mathcal{F}$ be a specification. (As mentioned earlier, formally these are just two protocols, and the different names are meant only to help the intuition.) Also let $\rho = \rho[\pi]$ be a protocol that includes subroutine calls to π. Below we denote by ρ^{π} the system where the subroutine calls to π are actually processed by π and by $\rho^{\mathcal{F}/\pi}$ the system where these subroutine calls are processed by $\mathcal{F}$.

The UC theorem [Can01] states that if π UC-realizes $\mathcal{F}$, then ρ^{π} UC-realizes $\rho^{\mathcal{F}/\pi}$; however, that theorem does not hold in the presence of the modularity-breaking aggregator $\mathcal{G}$. The proof of the UC-theorem in [Can01] relies on all the processes being "modular"; namely, a process can only interact with its caller and its subroutines (and the adversary).[4]

As we have seen, modularity is incompatible with the definition of leaky protocols; indeed, all processes are required to interact with the aggregator, which is neither their caller nor their subroutine (nor an adversarial entity). Still, if π realizes $\mathcal{F}$ with a leakage-oblivious simulator, we can recover the same result. Below we call a protocol "modular up to leakage" if it only interacts with its caller, its subroutines, the adversary, and the aggregator.

Theorem 3.1 (UC-composition with leakage). *Let $\rho, \pi, \mathcal{F}$ be protocols as above, all modular up to leakage, such that π UC-emulates $\mathcal{F}$ with a leakage-oblivious simulator. Then ρ^{π} UC-emulates $\rho^{\mathcal{F}/\pi}$. Furthermore, it does so with a leakage-oblivious simulator.*

Proof overview. The proof follows the outline of the proof of the basic UC theorem; here, we focus on the required adjustments due the leakage. For sake of simplicity, in this overview we assume that ρ invokes only a single instance of the sub-protocol π.

[4] Such protocols are also called "subroutine respecting".

Recall that we need to construct a leakage-oblivious simulator $\mathcal{S}_\rho$ such that no environment can tell whether it is interacting with ρ^π and the $\mathsf{dummy\ adversary}$ $\mathcal{D}$, or with $\rho^{\mathcal{F}/\pi}$ and $\mathcal{S}_\rho$. The construction of $\mathcal{S}_\rho$ is naturally based on the leakage-oblivious simulator $\mathcal{S}_\pi$ as guaranteed by the premise. That is, $\mathcal{S}_\rho$ runs a copy of $\mathcal{S}_\pi$; as in the basic UC theorem, the interaction between $\mathcal{Z}$ and the parties is separated into two parts. The interaction with π is dealt with by $\mathcal{S}_\pi$, which generates messages for the corresponding sub-parties and handles incoming messages from these parties. The effect of the environment on rest of the system, is handled by direct interaction with the external parties running ρ.

Leakage queries are handled by way of a subroutine $\tilde{\mathcal{S}}_\rho$ that generates a state translation T_ρ, as needed for leakage-oblivious simulation. Recall that the leakage function L that $\mathcal{S}_\rho$ receives from the environment was designed to be applied to a "real protocol state" in ρ^π (and since ρ runs a single copy of π then this state is of the form $(\mathsf{state}_\rho, \mathsf{state}_\pi)$). The simulator $\mathcal{S}_\rho$, on the other hand, can only ask the aggregator for leakage on the state of $\rho^{\mathcal{F}}$, which is of the form $(\mathsf{state}_\rho, \mathsf{state}_{\mathcal{F}})$. To bridge this gap, $\tilde{\mathcal{S}}_\rho$ runs the "state translation subroutine" $\tilde{\mathcal{S}}_\pi$. (This can be done since $\mathcal{S}_\rho$ has the entire current state of $\mathcal{S}_\pi$.) Once $\tilde{\mathcal{S}}_\pi$ produces a state translation function T_π, $\tilde{\mathcal{S}}_\rho$ generates its own state translation function $T_\rho\left(\mathsf{state}_\rho, \mathsf{state}_{\mathcal{F}}\right) = \left(\mathsf{state}_\rho, T_\pi\left(\mathsf{state}_{\mathcal{F}}\right)\right)$ and sends to the aggregator a leakage function L', where

$$L'\left(\mathsf{state}_\rho, \mathsf{state}_{\mathcal{F}}\right) \;=\; L\left(T_\rho\left(\mathsf{state}_\rho, \mathsf{state}_{\mathcal{F}}\right)\right) \;=\; L\left(\mathsf{state}_\rho, T_\pi\left(\mathsf{state}_{\mathcal{F}}\right)\right) \quad .$$

Observe that already at this stage we rely crucially on $\mathcal{S}_\pi$ being leakage-oblivious: if $\mathcal{S}_\pi$ was expecting to see a leakage function $L_\pi\left(\mathsf{state}_\pi\right)$ before producing the translation, then we could not use it (since $\mathcal{S}_\rho$ does not know the state state_ρ, and therefore cannot write the description of the induced function $L_{\mathsf{state}_\rho}\left(\mathsf{state}_\pi\right) = L\left(\mathsf{state}_\rho, \mathsf{state}_\pi\right)$). Once the aggregator returns an answer, $\mathcal{S}_\rho$ passes it to the environment and returns to its previous state (including the previous state of the sub-simulator $\mathcal{S}_\pi$).

It is clear from the description that $\mathcal{S}_\rho$ is leakage-oblivious. The validity of $\mathcal{S}_\rho$ is shown by reduction to the validity of $\mathcal{S}_\pi$. That is, given an environment $\mathcal{Z}_\rho$ that distinguishes an execution of $(\rho^\pi, \mathcal{D})$ from an execution of $(\rho^{\mathcal{F}/\pi}, \mathcal{S}_\rho)$, we construct an environment $\mathcal{Z}_\pi$ that distinguishes an execution of $(\pi, \mathcal{D})$ from an execution of $(\mathcal{F}, \mathcal{S}_\pi)$. The environment $\mathcal{Z}_\pi$ simulates an execution of $(\mathcal{Z}_\rho, \rho)$ "in its head", except that all messages corresponding to π are forwarded to the external execution. Indeed, leakage queries aside, we have: (a) if the external execution consists of $\mathcal{S}_\pi$ and $\mathcal{F}$, then the entire (composed) execution amounts to running $\mathcal{Z}_\rho$ with $\mathcal{S}_\rho$ and $\rho^{\mathcal{F}}$; (b) if the external execution consists of $\mathcal{D}$ and π, then the entire (composed) execution amounts to running $\mathcal{Z}_\rho$ with $\mathcal{D}$ and ρ^π.

Extending this argument to include leakage, the environment $\mathcal{Z}_\pi$ acts as follows. When $\mathcal{Z}_\rho$ produces a leakage query L to be evaluated on $\mathsf{state}_\rho, \mathsf{state}_\pi$, $\mathcal{Z}_\pi$ computes the simulated state state_ρ and computes the restricted leakage function $L_{\mathsf{state}_\rho}\left(\mathsf{state}_\pi\right) = L\left(\mathsf{state}_\rho, \mathsf{state}_\pi\right)$, which should be evaluated only on state_π. Note that since $\mathcal{S}_\pi$ is leakage-oblivious, the state-translation function that it outputs when run as a subroutine of $\mathcal{S}_\rho$ is the same as the state-translation function that it outputs when run with the environment $\mathcal{Z}_\pi$. The rest of the argument remains unchanged.

The actual proof also deals with the case where multiple instances of the subroutine π are invoked and can be found in the full version of this work [BCH11].

4 From Adaptive Security to Leakage-Tolerance

Recall that the adversary in the UC framework can adaptively corrupt parties during protocol execution, thereby learning their entire internal state. If the corruption is passive (semi-honest), the party keeps following the same program as it did before the corruption, and if it is Byzantine (malicious), then the adversary also gains control of the program that the party runs from now on.

As already pointed out, leakage can be thought of as a form of corruption, where the adversary gains partial information on the inner state of a party. The converse is also true, passive corruption can be viewed simply as leaking the entire internal state. The challenges in simulation are also similar: for both corruption and leakage the simulator must translate some "ideal state" that it gets from the functionality into a "real state" that it can show the environment, and do it in a way that is consistent with the transcript so far. Below we formalize this similarity, showing that "in principle" a protocol that realizes some functionality $\mathcal{F}$ in the presence of passive adaptive corruptions also realizes it in the presence of leakage. There are considerable restrictions, however. Most importantly, the implication holds only for corruption-oblivious simulators (see below). Also, $\mathcal{F}$ must be adapted to handle leakage queries, and we prove the implication for a particular (natural) way of doing this adaptation.

Adapting functionalities to leakage. Let $\mathcal{F}$ be functionality that was designed for a leakage-free model with corruptions. This means that $\mathcal{F}$ already has some mechanism to reply to messages from the adversary about corruptions of players. We now need to adapt it by explaining how it reacts to leakage queries from the aggregator $\mathcal{G}$. The adaptation is natural: whenever $\mathcal{G}$ asks for the state of party pid for the purpose of leakage, the functionality replies with exactly the same thing that it would have given the adversary if pid was passively corrupted at this time. Then, once $\mathcal{G}$ reports the number of leakage bits, the functionality forwards this number on the I/O lines of party pid.[5] Thereafter, the functionality behaves just as if party pid was passively corrupted. We denote the resulting functionality by $\mathcal{F}^{+\mathsf{lk}}$. We stress that if $\mathcal{F}$ was designed to react differently to passive and Byzantine corruptions, then it uses the passive corruption mode to handle leakage.

Note the implication of viewing leakage as corruption: in principle, reaction to leakage could be gradual - a functionality $\mathcal{F}$ can change its behavior proportionally to the amount of leakage, or to have a leakage threshold up to which it does one thing and after which it does another. However, the reaction of $\mathcal{F}$ to

[5] This number-reporting action is meant to allow the environment to do its leakage bookkeeping, and for ideal functionalities to be able to react to leakage.

(passive) corruption is typically "all or nothing", it is either not affected or it completely "gives up". Using our convention from above, this "all or nothing" reaction is carried over to $\mathcal{F}^{+\mathsf{lk}}$. For example, if $\mathcal{F}$ is an authenticated channels functionality, then $\mathcal{F}^{+\mathsf{lk}}$ will permit forgery as soon as even a single bit is leaked. On the other hand, if $\mathcal{F}$ is a commitment functionality then leakage events have no effect on the subsequent behavior of $\mathcal{F}^{+\mathsf{lk}}$. Although the transformation that we prove below works for every functionality $\mathcal{F}$, its usefulness depends crucially on the way $\mathcal{F}$ handles passive corruptions.

Corruption-oblivious simulators. The intuition for why adaptive corruption implies leakage-tolerance is that if we can simulate the **entire** state of an adaptively corrupted party, then we should also be able to simulate only parts of its state (according to a particular leakage function). The problem with this intuition, however, is that future behavior of the simulator may depend on the entire state learned during corruption, which is not available to the leakage simulator.

We thus restrict our attention to special simulators that are oblivious of the state learned during corruption (similarly to the leakage-oblivious simulators from Section 2). As for leakage, we only define corruption-oblivious simulators for the dummy adversary $\mathcal{D}$ (which is sufficient). The simulator $\mathcal{S}$ for $\mathcal{D}$ should have a special subroutine $\tilde{S}$ for handling passive corruptions. When $\mathcal{S}$ receives from the environment a request (passive corrupt, pid) to passively corrupt a party pid, $\mathcal{S}$ invokes $\tilde{S}$ to produce a state translation function T. T is used to transform the "internal state" that $\mathcal{F}$ (or the hybrid-world protocol) returns for party pid into a state of the "real world" implementation protocol π for this party. Then, $\mathcal{S}$ sends a passive corrupt message to pid, obtains the corresponding state (from $\mathcal{F}$ or the hybrid-world instance), applies to it the transformation T and returns the result $\mathsf{state}_\pi = T(\mathsf{state})$ to the environment. After the result is forwarded to the environment, $\mathcal{S}$ returns to its state prior to the time it invoked $\tilde{S}$.

Note that since this is passive corruption, then party pid can keep evolving its state after the initial corruption, and the environment can ask to see the updated state from time to time. $\mathcal{S}$ handles each such update request as a new passive corruption query, invoking $\tilde{S}$ again to get state-translation function, calling the functionality again, etc. (We note that there is no restriction on the way that $\mathcal{S}$ handles Byzantine corruptions.)

We stress that $\mathcal{S}$ does not make any direct use of the state of the corrupted parties. In particular, the future operation of $\mathcal{S}$, when simulating the messages generated by corrupted parties, is done independently of their secret local states. As seen in subsequent sections, in some cases this turns out to be a strong restriction (see Example 5.1 in Section 5). We are now ready to state the main result of this section. The proof is provided in the full version [BCH11].

Theorem 4.1. *Let π be a protocol that UC-realizes an ideal functionality $\mathcal{F}$ in the presence of passive adaptive corruptions (but no leakage), with a corruption-oblivious simulator. Then π also UC-realizes $\mathcal{F}^{+\mathsf{lk}}$ with a leakage-oblivious simulator in the UC model with leakage.*

Composition of corruption-oblivious simulators. We note that, viewing corruption-oblivious simulators as a special case of leakage-oblivious simulators (for leaking the identity function), the proof of the leaky UC Theorem 3.1 implies that corruption-oblivious simulation is preserved under universal composition:

Corollary 4.1 (of Theorem 3.1). *Let $\rho, \pi, \mathcal{F}$ be protocols that are modular up to leakage, such that π UC-emulates $\mathcal{F}$ with a corruption-oblivious simulator. Then ρ^{π} UC-emulates $\rho^{\mathcal{F}/\pi}$ with a corruption-oblivious simulator.*

5 Realizing Leaky Adaptations of Basic Interactive Tasks

This section describes the construction of leakage-tolerant protocols for for several interactive tasks. We describe constructions for secure message transmission, (semi-honest) oblivious transfer, commitment and zero-knowledge. These constructions all assume ideal authenticated channels. We then present a construction of leakage-resilient authenticated channels. All of our constructions are composable. We conclude the section with a discussion on the difficulties in obtaining general leakage-tolerant multi party computation.

The bulk of this section is omitted. It can be found in the full version of this work [BCH11]. Here, we only sketch the constructions for the last two tasks.

5.1 Zero-knowledge from Ideal Leaky Commitments

We adapt the zero-knowledge ideal functionality to tolerate leakage and demonstrate a protocol that realizes the adapted functionality in the presence of leakage. Recall that $\mathcal{F}_{\mathsf{ZK}:R}$, for a relation R, takes from the prover P an input (x, w), and outputs x to the verifier V only if $R(x, w)$ holds. This formulation guarantees to P perfect secrecy of w. It also guarantees perfect soundness to V.

Adapting $\mathcal{F}_{\mathsf{ZK}}$ to leakage, we can ideally hope to realize a functionality with optimal tolerance, such as $\mathcal{F}_{\mathsf{ZK}}^{+\mathsf{lk}}$, which can "gracefully" tolerate arbitrary leakage from the prover, and in addition does not give up on soundness even in face of arbitrary leakage on the verifier. However, we could not manage to realize such a functionality. Instead, we consider an adaptation that can tolerate arbitrary leakage from the prover, but only a bounded amount of leakage from the verifier before soundness breaks. Before presenting our eventual adaptation and implementation, we briefly sketch the difficulties which prevent us from achieving optimal leakage-tolerance.

As shown in [CF01, CLOS02], the parallel repetition of classic 3-round zero-knowledge protocols, such as Blum's Hamiltonian cycle [Blu86], and GMW's 3-coloring [GMW91], UC-realizes the basic (non-leaky) $\mathcal{F}_{\mathsf{ZK}}$, given access to (non-leaky) ideal commitment. Moreover, they do so even in the presence of adaptive corruptions. However, the proofs of security of these protocols do *not* yield corruption-oblivious simulation. Thus, we cannot conclude that these protocols UC-realize $\mathcal{F}_{\mathsf{ZK}}^{+\mathsf{lk}}$ under leakage.

In fact, without any modifications, these protocols seem inherently impossible to simulate in the face of leakage. To demonstrate this, let us recall GMW's

3-coloring protocol. Here, the prover, who possesses a 3-coloring c, chooses a random permutation σ of the three colors and commits to the permuted coloring $\sigma(c)$. The verifier then requires that the prover opens the colors of a random edge and checks that its endpoints are indeed colored differently. Now, consider a (Byzantinely) corrupted verifier V^* that also obtains leakage on the prover's coloring during the protocol. This verifier can leak, for example, the secret permutation σ and then ask the (honest) prover to open the colors $\sigma(c(i)), \sigma(c(j))$ of some random edge (i, j). Finally, it can leak again the true colors $c(i), c(j)$. Simulating such a behavior seems impossible (assuming 3COL $\notin$ BPP). Indeed, once the simulator simulates σ for the first leakage, it essentially becomes committed to it for the rest of the protocol. Then, when it is required to simulate the opening of $\sigma(c(i)), \sigma(c(j))$, it essentially has no information on c, and hence, if it can consistently simulate the second leakage query, then essentially it must "know" a proper coloring of the entire graph.[6] We stress that this inherent difficulty also fails simulators that are not leakage-oblivious (and are thus allowed to depend on both the leakage function and the leakage-result).

To overcome the above problem, we require that at the beginning of the protocol, the verifier commits to all its challenges. This already allows the simulation to go through; now the simulator can first extract the challenge edge (i, j), choose random colors for it $c'(i), c'(j)$, and then have the leakage return a permutation mapping the real $c(i), c(j)$ to $c'(i), c'(j)$. In fact, we show that this adjustment is enough for simulating any malicious verifier.

This adjustment comes, however, at a price: unlike the original protocols, where the verifier was of the public coins type (and had no secret state), now the verifier commits to its challenges, and the secrecy of these challenges is crucial for the protocol's soundness. Hence, we cannot hope that in such a protocol the verifier will be able to withstand arbitrary amounts of leakage; in particular, once the prover leaks all of the verifier's challenge, soundness is doomed.

Consequently, we only realize a weaker adaptation, where the verifier can only tolerate a bounded amount of leakage. (The prover can still tolerate arbitrary leakage.) More specifically, we can tolerate arbitrary leakage on the verifier's randomness so long that a super-logarithmic amount of min-entropy is maintained.

5.2 Authenticated Channels

We construct a protocol for realizing *leaky authenticated channels* with bounded leakage-resilience. More specifically, the protocol UC-realizes an ideal functionality $\mathcal{F}_{\mathsf{Auth}}^{+B}$ that guarantees authenticated communication *as long as the overall leakage between any two transmissions of some messages does not exceed a prespecified bound B.*

The protocol we present uses two main building blocks: (a) non-committing encryption (NCE) (b) information theoretic c-time message authentication codes (MACs) that are resilient to a constant leakage rate from the secret key. The

[6] This intuition can be made formal; namely, given such a simulator we can construct an algorithm for 3-coloring arbitrary graphs.

idea behind the protocol is simple. The parties initially share a (leaky) secret key K_1. Then the protocol proceeds inductively; at each round, a current authentication key K_i is used to authenticate the i-th message, m_i. In addition, a fresh key K_{i+1} is generated and transmitted using non-committing encryption. These transmitted ciphers are also authenticated using K_i. To allow the authentication to go through, we need our underlying leaky MAC scheme to allow authentication of messages that are polynomially longer than the secret key. This is achieved using *universal hashing*. Concretely, the protocol we present tolerates, between each two transmissions, roughly $k/10$ bits of leakage on the k-long secret key. Similar techniques are used for a related goal in [BCG$^+$11]. However, the security analysis there is different than the one here.

The protocol we construct admits leakage-oblivious simulation and is thus composable. We can, therefore, use it as a basic building block supporting any protocol that requires authenticated channels, when ideally authenticated channels are unavailable. We stress, however, that when doing so the leakage-tolerance of the higher-level protocol, naturally degrades to that of the authentication protocol.

5.3 On the Difficulty in Achieving General Leakage-Tolerant MPC

Equipped with Theorem 4.1, we may hope that, similarly to the tasks considered above, general leakage-tolerant multi-party computation (MPC) would also follow from known results on adaptively secure MPC (such as, [CLOS02]). Unfortunately, known results do not admit corruption-oblivious simulation and are in fact far from being leakage-tolerant. We exemplify the relevant difficulties by giving a protocol that is adaptively secure but not leakage-tolerant. Although seemingly contrived, the protocol suffers from the same caveats that fail known adaptively secure protocols from achieving leakage-tolerance.

Example 5.1. Let $\mathcal{F}$ be a standard corruption functionality that takes n-bit inputs from two parties, P_0 and P_1, and outputs nothing. As soon as party P_i provides input x_i, the virtual local state of P_i is set to x_i. Now, consider the following protocol π: first, the parties give their inputs to some trusted party that returns a random b_i to P_i such that $b_0 + b_1 = \langle x_0, x_1 \rangle$ where $\langle , \rangle$ denotes inner-product in $\mathbb{F}_2$. (The inner product can be replaced by any two-source extractor.) Next, the parties output nothing and halt.

It can be seen that π securely realizes $\mathcal{F}$ with respect to adaptive corruptions. This is so since, once the first party P_i is corrupted, the simulator learns x_i and can give x_i to the adversary, plus a random bit instead of b_i. Now, when P_{1-i} is corrupted, the simulator learns x_{1-i} and can determine the bit b_{1-i} so that $b_0 + b_1 = \langle x_0, x_1 \rangle$. However, notice that here the simulator is not corruption-oblivious: the handling of the second corruption depends on the input value x_i of the first corrupted party. Indeed, π does not realize $\mathcal{F}^{+\mathsf{lk}}$ with even one bit of leakage from each party: the adversary can ask to leak b_i from P_i and thus learn $\langle x_0, x_1 \rangle$. However, in the ideal model for $\mathcal{F}^{+\mathsf{lk}}$, assuming x_0, x_1 are long random strings, the simulator has no hope of learning $\langle x_0, x_1 \rangle$. This is so since in the ideal model, the simulator can only perform one-bit leakage on x_0 and x_1 separately, and hence it can not guess $\langle x_0, x_1 \rangle$ with non-negligible advantage.

Indeed, the same problem would arise in GMW-based protocols, where the value of each wire is secret-shared between the parties in a non leakage-resilient manner as above. This is actually also the case for YAO-based adaptively secure protocols (for NC_1 functions); there also (although not explicitly), the value of each wire is effectively secret-shared between the parties in a non leakage-resilient way.

Weak (joint-state) leakage-tolerance Vs. Strong (separate-state) leakage-tolerance. Note that, had we modified $\mathcal{F}^{+\mathsf{lk}}$ in the above example so that the virtual local state of each party includes both inputs, the above protocol would UC-realize $\mathcal{F}^{+\mathsf{lk}}$ with leakage. More generally, if we settle for a weaker leakage-tolerance guarantee where the ideal world simulator can jointly leak from the inputs and outputs of all parties (and not only separately from the inputs and outputs of each leaking party alone), then leakage-tolerance can already be achieved. In fact, combining our leakage-tolerant OT protocol with an adaptively secure protocol, such as GMW, it is easy to obtain semi-honest MPC for general functions. (We note that this, in particular, concerns the two party setting where one party is statically corrupted considered by [DHP11], which can be seen as a special case of weak leakage-tolerance.)

However, in a setting where real world adversaries are restricted to separate leakage from each party, an ideal process that allows joint leakage from the internal states of the parties is somewhat unsatisfactory. Achieving strong (separate-state) leakage-tolerant MPC in general (without preprocessing or limitations on the number of honest parties) remains an interesting open question.

Acknowledgments. We thank Amit Sahai for telling us about the problems with proving leakage-tolerance of the standard three round zero-knowledge protocols and about the way this problem is solved in [GJS11].

References

[ADN$^+$10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs, *Public-key encryption in the bounded-retrieval model*, EUROCRYPT, 2010, pp. 113–134.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs, *Survey: Leakage Resilience and the Bounded Retrieval Model*, Information Theoretic Security - ICITS 2009 (Kaoru Kurosawa, ed.), 2009, pp. 1–18.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan, *Simultaneous hardcore bits and cryptography against memory attacks*, TCC, 2009, pp. 474–495.

[BCG$^+$11] Nir Bitansky, Ran Canetti, Shafi Goldwasser, Shai Halevi, Yael Tauman Kalai, and Guy N. Rothblum, *Program obfuscation with leaky hardware*, ASIACRYPT, 2011, pp. 722–739.

[BCH11] Nir Bitansky, Ran Canetti, and Shai Halevi, *Leakage-tolerant interactive protocols*, 2011, Full version available at `http://eprint.iacr.org/2011/204`.

[BGK11] Elette Boyle, Shafi Goldwasser, and Yael Tauman Kalai, *Leakage-resilient coin tossing*, DISC, 2011, available at `http://eprint.iacr.org/2011/291`, pp. 181–196.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan, *Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage*, FOCS, 2010, pp. 501–510.

[Blu86] Manuel Blum, *How to prove a theorem so no one else can claim it*, 1986, International Congress of Mathematicians, pp. 444–451.

[BSW11] Elette Boyle, Gil Segev, and Daniel Wichs, *Fully leakage-resilient signatures*, EUROCRYPT, 2011, pp. 89–108.

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

[CF01] Ran Canetti and Marc Fischlin, *Universally composable commitments*, CRYPTO, vol. 2139, 2001, pp. 19–40.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor, *Adaptively secure multi-party computation*, STOC, 1996, pp. 639–648.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai, *Universally composable two-party and multi-party secure computation*, STOC, 2002, pp. 494–503.

[DHLAW10a] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs, *Cryptography against continuous memory attacks*, FOCS, 2010, pp. 511–520.

[DHLAW10b] ———, *Efficient public-key cryptography in the presence of key leakage*, ASIACRYPT, 2010, pp. 613–631.

[DHP11] Ivan Damgård, Carmit Hazay, and Arpita Patra, *Leakage resilient secure two-party computation*, IACR Cryptology ePrint Archive (2011), 256, `http://eprint.iacr.org/2011/256`.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett, *On cryptography with auxiliary input*, STOC, 2009, pp. 621–630.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak, *Leakage-resilient cryptography*, FOCS, IEEE Computer Society, 2008, pp. 293–302.

[GJS11] Sanjam Garg, Abhishek Jain, and Amit Sahai, *Leakage-resilient zero knowledge*, CRYPTO, 2011, pp. 297–315.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson, *Proofs that yield nothing but their validity for all languages in np have zero-knowledge proof systems*, J. ACM **38** (1991), no. 3, 691–729.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai, *Non-interactive zaps and new techniques for nizk*, CRYPTO, 2006, pp. 97–111.

[HL11] Shai Halevi and Huijia Lin, *After-the-fact leakage in public-key encryption*, TCC (Yuval Ishai, ed.), 2011, pp. 107–124.

[MR04] Silvio Micali and Leonid Reyzin, *Physically observable cryptography*, TCC, 2004, pp. 278–296.

[NS09] Moni Naor and Gil Segev, *Public-key cryptosystems resilient to key leakage*, CRYPTO, 2009, pp. 18–35.

[Pie09] Krzysztof Pietrzak, *A leakage-resilient mode of operation*, EUROCRYPT, 2009, pp. 462–482.

[Sta09] Francois-Xavier Standaert, *Introduction to side-channel attacks*, Secure Integrated Circuits and Systems (Ingrid M.R. Verbauwhede, ed.), Springer, 2009, pp. 27–44.