David and Goliath Commitments: UC Computation for Asymmetric Parties Using Tamper-Proof Hardware

Tal Moran and Gil Segev

Department of Computer Science and Applied Mathematics,
Weizmann Institute of Science, Rehovot 76100, Israel.
Email: {tal.moran,gil.segev}@weizmann.ac.il

Abstract. Designing secure protocols in the Universal Composability (UC) framework confers many advantages. In particular, it allows the protocols to be securely used as building blocks in more complex protocols, and assists in understanding their security properties. Unfortunately, most existing models in which universally composable computation is possible (for useful functionalities) require a trusted setup stage. Recently, Katz [Eurocrypt '07] proposed an alternative to the trusted setup assumption: tamper-proof hardware. Instead of trusting a third party to correctly generate the setup information, each party can create its own hardware tokens, which it sends to the other parties. Each party is only required to trust that its own tokens are tamper-proof.
Katz designed a UC commitment protocol that requires both parties to generate hardware tokens. In addition, his protocol relies on a specific number-theoretic assumption. In this paper, we construct UC commitment protocols for "David" and "Goliath": we only require a single party (Goliath) to be capable of generating tokens. We construct a version of the protocol that is secure for computationally unbounded parties, and a more efficient version that makes computational assumptions only about David (we require only the existence of a one-way function). Our protocols are simple enough to be performed by hand on David's side.
These properties may allow such protocols to be used in situations which are inherently asymmetric in real-life, especially those involving individuals versus large organizations. Classic examples include voting protocols (voters versus "the government") and protocols involving private medical data (patients versus insurance-agencies or hospitals).

1 Introduction

Designing secure protocols that run in complex environments, such as those typically found in real-world applications, is a very challenging task. The design must take into account that such protocols may be executed concurrently with multiple other copies of the same protocol (e.g., many voters voting at the same time) or with different protocols (e.g., performing an electronic bank transaction in response to the results of an on-line auction). The *Universal Composability* (UC) framework was introduced by Canetti [5] to model the security of cryptographic protocols when executed in such environments. Protocols proven secure within the UC framework are, in particular, secure even under arbitrary composition.

Unfortunately, it turns out that unless a majority of the participating parties are honest (which can never be assumed when there are only two participants), almost no useful functionality can be securely realized in this framework [7, 8]. On the positive side, however, Canetti and Fischlin [7] managed to circumvent these impossibility results by assuming a strong form of setup – a common reference string (CRS). This suffices for realizing any (well-formed) functionality in the UC framework while tolerating any number of dishonest parties [9].

The main drawback of assuming the availability of a CRS is that this requires trust in the party that constructs the CRS, and there are no security guarantees if the CRS is set in an adversarial manner. This state of affairs motivated the research of alternative setup assumptions that can circumvent the impossibility results, and imply the feasibility of securely realizing natural functionalities in the UC framework. A variety of alternative setup assumptions have already been explored, such as public-key registration services [1, 6], signature cards [19] and a variation of the CRS assumption in which multiple strings are available [18].

The above mentioned setup assumptions still require trust in at least some parties in the system. Recently, Katz [20] proposed an alternative assumption that eliminates the need for such trusted parties. Katz suggested basing UC computations on a physical assumption: the existence of tamper-proof hardware. Under this assumption, an honest party can construct a hardware token T_F implementing any polynomial-time functionality F, but an adversary given the token T_F can do no more than observe the input/output characteristics of this token. Katz showed that such a primitive can be used, together with standard cryptographic assumptions, to realize the ideal multiple commitment functionality in the UC framework while tolerating any number of dishonest parties, and hence to realize general UC multi-party computation [9].

1.1 Our Contributions

We revisit Katz's proposal of basing universally composable computations on tamper-proof hardware. More specifically, we focus on realizing the ideal commitment functionalities (which suffice for realizing general UC multi-party computation [9]). In this work, we construct UC commitment protocols using tamper-proof hardware tokens that have several advantages over Katz's protocol:

David and Goliath: asymmetric assumptions suffice. Katz's commitment protocol is symmetric with respect to the assumptions about the two parties: both parties must create a hardware token, and both must assume that their token is proof against tampering by the other party. In many situations, however, the two participating parties are not symmetric. For example, in a voting scenario, voters may not be able to create their own hardware, or may not trust that hardware they create (or buy) is truly "tamper-proof" against the government (the other party in a voting protocol).

Our commitment protocols only require a single party (Goliath) to generate a token. The other party (David) must ensure the token cannot communicate with Goliath, but does not have to make any assumptions about the power of Goliath. We construct different commitment protocols for Goliath as the sender and for Goliath as the receiver.

Reducing the computational assumptions. In addition to relying on the existence of tamper-proof hardware tokens, Katz's protocol relies on a specific number-theoretic assumption — the decisional Diffie-Hellman assumption. Katz posed as an open problem to rely on general computational assumptions (under the assumption that tamper-proof hardware exists). We answer this open problem and reduce the required computational assumptions. Our contributions in this regard are as follows:

- We demonstrate that computational assumptions are not necessary in order to realize the ideal functionality $\mathcal{F}_{\mathsf{COM}}$ (this functionality allows a single commitment for each hardware token) if we assume the existence of tamper-proof hardware tokens. That is, our protocols that realize $\mathcal{F}_{\mathsf{COM}}$ do not rely on any computational assumptions. These protocols are also secure against adaptive adversaries (although the proof for the adaptive case is deferred to the full version of the paper).
- We demonstrate that the existence of one-way functions suffices in order to realize the ideal functionality $\mathcal{F}_{\mathsf{MCOM}}$ (this functionality allows a polynomial number of concurrent commitments using the same hardware token) if we also assume the existence of tamper-proof hardware tokens.
- In keeping with the David and Goliath theme, even the protocols based on one-way functions do not make assumptions about Goliath's computational power.

"Bare-handed" protocols. The protocols presented in this paper are highly efficient, and in particular require David to perform only a few (elementary and simple) operations, such as comparing two bit-strings or computing the exclusive-or of two bit-strings (even in the protocol based on one-way functions). When David is the receiver, these operations can be performed by "bare-handed" humans without the aid of computers (when David is the sender, he may require a calculator). Such a property is useful in many situations where computers cannot be trusted or where "transparency" to humans is essential (which is the case, for example, when designing voting protocols). In addition, the same efficiency guarantees hold for the hardware tokens in the protocols that are secure against computationally unbounded adversaries; such tokens may be constructed using extremely constrained devices.

1.2 Related Work

Basing cryptographic protocols on physical assumptions. Basing cryptographic protocols on physical assumptions is a common practice. Perhaps the most striking example is the field of quantum cryptography, where the physics of quantum mechanics are used to implement cryptographic operations – some of which are impossible in the "bare" model. For example, Bennett and Brassard [2] achieved information-theoretically secure key agreement over public channels based only on assumptions about the physics of quantum mechanics.

Much work has been done on basing commitment schemes and oblivious transfer protocols on the physical properties of communication channels, using the random noise in a communication channel as the basis for security. Both commitment schemes and oblivious transfer protocols were shown to be realizable in the *Binary Symmetric Channel* model [13, 14], in which random noise is added to the channel in both directions with some known probability. Later works show that they can also be implemented, under certain conditions, in the weaker (but more convincing) *Unfair Noisy Channel* model [15, 16], where the error probability is not known exactly to the honest parties, and furthermore can be influenced by the adversary.

The work of Katz [20] was inspired by works of Chaum and Pedersen [11], Brands [4], and Cramer and Pedersen [12] that proposed the use of smartcards in the context of e-cash. The reader is referred to [20] for a brief description of their approach. More recently, Moran and Naor [21] demonstrated the possibility of implementing oblivious transfer, bit-commitment and coin flipping based on "tamper-evident seals" that model very intuitive physical models: sealed envelopes and locked boxes.

Concurrent independent work. Independent of this paper, Chandran, Goyal and Sahai [10] and Damgård, Nielsen and Wichs [17] address the problem of basing universally composable computations on tamper-proof hardware, and construct protocols realizing the ideal multiple commitment functionality $\mathcal{F}_{\mathsf{MCOM}}$.

Chandran et al. [10] show that $\mathcal{F}_{\mathsf{MCOM}}$ can be realized based on tamper-proof hardware tokens and one-way functions. The main advantage of their approach is that their security proof does not rely on the simulator's ability to rewind hardware tokens. This gives their protocol security against *reset* attacks (where the adversary can rewind the tokens). In particular, their protocol does not require hardware tokens to keep state between invocations. In addition, their protocol does away with the requirement that the parties know the code of the token which they distribute (we note that this assumption is essential both to Katz's construction and to ours).

Damgård et al. [17] focus on relaxing the "isolation" requirement and allow the hardware tokens to communicate with the outside world as long as the number of communicated bits in both direction is below some pre-determined threshold (which is polynomial in the security parameter). With this relaxation in mind, they realize $\mathcal{F}_{\mathsf{MCOM}}$ assuming the existence of one-way permutations and a semantically-secure dense public-key encryption scheme with pseudorandom ciphertexts.

The main advantage of our work over those of Chandran et al. [10] and Damgård et al. [17] is that in our constructions only one of the parties is required to create its own hardware token. As argued above, this is desirable and often essential in many scenarios. The constructions of Chandran et al. and Damgård et al. require that each party creates its own hardware token. That is, it is assumed that both parties have the resources required to create hardware tokens.

1.3 Paper Organization

The remainder of this paper is organized as follows. For those not familiar with the UC framework, we give some background in Section 2, as well as formal definitions of the different commitment functionalities. In Section 3, we briefly review the formal model for tamper-proof hardware tokens. In Section 4, we describe our protocols for UC commitment where Goliath is the sender, and sketch their proof of security. Section 5 does the same for the protocol in which David is the sender. Finally, Section 6 contains a discussion and some open problems.

2 The UC Commitment Functionalities

Many two-party functionalities can be easily implemented in a natural "secure" manner using a trusted third party that follows pre-agreed rules. In proving that a two-party protocol is secure, it is highly desirable to argue that the protocol behaves "as if it was performed using the trusted third party". The Universally Composability (UC) framework is a formalization of this idea. In the UC framework, the trusted third party is called the *ideal functionality.* The ideal functionality is described by an interactive Turing machine that can communicate by authenticated, private channels with the participants of the protocol. We refer the reader to [5] for a more detailed exposition.

In the UC framework two ideal commitment functionalities are considered: functionality $\mathcal{F}_{\mathsf{COM}}$ that handles a single commitment-decommitment process, and functionality $\mathcal{F}_{\mathsf{MCOM}}$ that handles multiple such processes. The advantage of $\mathcal{F}_{\mathsf{MCOM}}$ over $\mathcal{F}_{\mathsf{COM}}$ in our setting is that protocols that securely realize $\mathcal{F}_{\mathsf{MCOM}}$ may use the same hardware token for multiple commitments.

In this paper we consider an additional ideal commitment functionality, one that handles a bounded number of commitment-decommitment processes. We refer to this functionality as the ideal bounded commitment functionality $\mathcal{F}_{\mathsf{BCOM}}$. Formal descriptions of $\mathcal{F}_{\mathsf{COM}}$, $\mathcal{F}_{\mathsf{BCOM}}$ and $\mathcal{F}_{\mathsf{MCOM}}$ are provided in Figures 1, 2 and 3, respectively.

Functionality $\mathcal{F}_{\mathsf{COM}}$

1. Upon receiving $(\mathsf{commit}, sid, P, P', b)$ from P, where $b \in \{0, 1\}$, record the value b and send $(\mathsf{receipt}, sid, P, P')$ to P' and the adversary. Ignore any subsequent commit messages.
2. Upon receiving $(\mathsf{open}, sid, P, P')$ from P, if some value b was previously recorded then send $(\mathsf{open}, sid, P, P', b)$ to P' and the adversary and halt. Otherwise halt.

Figure 1: The ideal commitment functionality.

Functionality $\mathcal{F}_{\mathsf{BCOM}}$

$\mathcal{F}_{\mathsf{BCOM}}$ is parameterized by a bound n on the number of allowed commitment-decommitment processes and an implicit security parameter k, and stores an internal counter j initialized to 0.

1. Upon receiving $(\mathsf{commit}, sid, cid, P, P', b)$ from P, where $b \in \{0,1\}$, if $j < n$ then set $j \leftarrow j+1$, record (cid, P, P', b) and send $(\mathsf{receipt}, sid, cid, P, P')$ to P' and the adversary. Ignore any subsequent $(\mathsf{commit}, sid, cid, P, P', \star)$ messages.
2. Upon receiving $(\mathsf{open}, sid, cid, P, P')$ from P, if some tuple (cid, P, P', b) was previously recorded then send $(\mathsf{open}, sid, cid, P, P', b)$ to P' and the adversary. Otherwise halt.

Figure 2: The ideal bounded commitment functionality.

Functionality $\mathcal{F}_{\mathsf{MCOM}}$

1. Upon receiving $(\mathsf{commit}, sid, cid, P, P', b)$ from P, where $b \in \{0,1\}$, record (cid, P, P', b) and send $(\mathsf{receipt}, sid, cid, P, P')$ to P' and the adversary. Ignore any subsequent $(\mathsf{commit}, sid, cid, P, P', \star)$ messages.
2. Upon receiving $(\mathsf{open}, sid, cid, P, P')$ from P, if some tuple (cid, P, P', b) was previously recorded then send $(\mathsf{open}, sid, cid, P, P', b)$ to P' and the adversary. Otherwise halt.

Figure 3: The ideal multiple commitment functionality.

3 Modeling Tamper-Proof Hardware

Our formulation of tamper-proof hardware tokens is based on the one provided by Katz [20]. Katz defined an ideal "wrapper" functionality, $\mathcal{F}_{\mathsf{WRAP}}$, which captures the intuitive idea that an honest party can construct a hardware token T_F implementing any polynomial-time functionality F, but an adversary given the token T_F can do no more than observe the input/output characteristics of this token. An honest party, given a token $T'_{F'}$ by an adversary, has no guarantee regarding the function F' that this token implements (other than what the honest user can deduce from the input/output of this device).

Figure 4 describes our formulation of the ideal functionality $\mathcal{F}_{\mathsf{WRAP}}$. Informally, a party P is allowed to create a hardware token, which is then delivered to P'. We refer to P as the creator of the token, and to P' as the user of the token. The hardware token encapsulates a Turing machine M which is provided by the creator P. At this point, the functionality allows the user P' to interact with the Turing machine M in a black-box manner. That is, P' is allowed to send messages of its choice to M via the wrapper functionality, and receive the corresponding answers.

As in Katz's formulation, we assume that the tokens are partially isolated, in the sense that a token cannot communicate with its creator. Our formulation also allows the tokens to maintain state between invocations. Although tokens created by honest parties are only required to maintain a limited state (such as the current round number), we allow tokens created by adversarial parties to maintain arbitrary state across invocations.

Functionality $\mathcal{F}_{\mathsf{WRAP}}$

$\mathcal{F}_{\mathsf{WRAP}}$ is parametrized by a polynomial $p(\cdot)$ and an implicit security parameter k.

1. Upon receiving $(\mathsf{create}, sid, P, P', M)$ from P, where M is a description of a Turing machine, do:
   (a) Send $(\mathsf{create}, sid, P, P')$ to P'.
   (b) If no tuple of the form $(P, P', \star, \star)$ is stored, then store $(P, P', M, \perp)$.
2. Upon receiving $(\mathsf{run}, sid, P, msg)$ from P', do:
   (a) If no tuple of the form $(P, P', \star, \star)$ is stored, then halt. Otherwise, retrieve the unique stored tuple $(P, P', M, state)$.
   (b) Run $M(msg, state)$ for at most $p(k)$ steps, and denote the result by $(out, state')$. If M does not respond in the allotted time then set $out = \perp$ and $state' = state$.
   (c) Send (sid, P, out) to P', store $(P, P', M, state')$ and erase $(P, P', M, state)$.

Figure 4: The ideal $\mathcal{F}_{\mathsf{WRAP}}$ functionality.

4 Constructing Goliath Commitments

In this section we describe protocols that realize the ideal *bounded commitment functionality*, $\mathcal{F}_{\mathsf{BCOM}}$ (see Figure 2), and the ideal *multiple commitment functionality*, $\mathcal{F}_{\mathsf{MCOM}}$ (see Figure 3). In these protocols, only the sender creates a hardware token (i.e., the sender is the powerful Goliath). Our protocol for realizing $\mathcal{F}_{\mathsf{BCOM}}$ does not rely on any computational assumptions, and our protocol for realizing $\mathcal{F}_{\mathsf{MCOM}}$ relies on the existence of any one-way function. In specifying the protocols we treat the hardware token as one of the parties in the protocol. The code executed by the token (i.e., the description of the Turing machine M sent to $\mathcal{F}_{\mathsf{WRAP}}$) is implicitly described by the token's role in the protocol.

Notation. For a bit m, we denote $\boldsymbol{m} \stackrel{def}{=} m \circ \cdots \circ m \in \{0,1\}^k$ the k-bit string consisting of k copies of m. We denote the bitwise complement of a string m by $\bar{m}$. The bitwise *xor* of two strings, a and b is denoted $a \oplus b$, while the bitwise *and* of a and b is denoted $a \odot b$.

4.1 Realizing the Ideal $\mathcal{F}_{\mathsf{BCOM}}$ Functionality

The intuition underlying our commitment protocol is that the hardware token is already a form of commitment — the sender is committing to a program that is hidden by the wrapping functionality, and cannot be changed once it is sent. The sender "hides" the comitted value in the program. The problem is that the receiver should not be able to extract the value before the opening phase. We solve this by using the fact that the token cannot communicate with the sender; the receiver sends the token a random challenge whose response depends on the hidden value, but does not reveal it. Because the sender does not know the challenge, he will be caught with high probability if he attempts to equivocate in the opening phase.

A formal description of the protocol is provided in Figure 5, which is followed by a sketch of its security proof.

Protocol $\mathsf{BCOM}_{\mathsf{Goliath}}$

Joint input: a security parameter k, and a bound n on the number of allowed commitment-decommitment processes.

Setup phase. The sender chooses n random pairs $(a_i, b_i) \xleftarrow{R} \{0,1\}^k \times \{0,1\}^k$ and creates a token with these parameters. The token also contains state in the form of a counter j initialized to 0. The sender sends the token to the receiver.

Commit phase. Denote the sender's input by (cid, m), and denote by i the number of invocations of the commit phase so far between the sender and the receiver. We assume w.l.o.g. that $cid = i$ (otherwise, both sides can maintain a database that translates between the two).

1. The sender computes $x_i \leftarrow a_i \oplus b_i \oplus \boldsymbol{m}$, and sends (i, x_i) to the receiver.
2. The receiver chooses a random challenge $c_i \xleftarrow{R} \{0,1\}^k$ and sends (i, c_i) to the token.
3. The token verifies that its internal counter $j = i$ and $j < n$ (otherwise it sends $\perp$ to the receiver and halts). It increments the counter $j \leftarrow j + 1$. The token computes $y_i \leftarrow a_i \odot c_i \oplus b_i \odot \bar{c_i}$ and sends (i, y_i) to the receiver (this is equivalent to letting each bit of c_i choose whether to send the corresponding bit of a_i or the corresponding bit of b_i).

Opening phase.

1. The sender sends m and (i, a_i, b_i) to the receiver.
2. The receiver computes $z_i \leftarrow a_i \oplus b_i$ and verifies that $z_i \oplus x_i = \boldsymbol{m}$ and that $a_i \odot c_i \oplus b_i \odot \bar{c_i} = y_i$. If not, it outputs $\perp$ and halts. Otherwise, it outputs m.

Figure 5: Protocol $\mathsf{BCOM}_{\mathsf{Goliath}}$.

Security intuition. To see why the protocol is hiding, note that after the commit phase the value $a \oplus b$ remains uniformly distributed from the receiver's point of view, regardless of the value of m (since, for every index ℓ, the receiver can choose to learn either the ℓ^{th} bit of a or the ℓ^{th} bit of b, but not both).

The protocol is binding because in order to equivocate, the sender must change at least $\frac{1}{2}k$ bits of a and b in the opening phase. Because the sender does not know the challenge sent to the token, and hence does not know which bits of a and b the receiver has already seen, if it tries to equivocate it will be caught with overwhelming probability.

Proof sketch. For simplicity we sketch the proof of security only for the case of a static adversary, as this case already captures the important ideas in the proof. A complete proof for the case of an adaptive adversary will be provided in the full version of this paper. In order to prove that the protocol realizes $\mathcal{F}_{\mathsf{BCOM}}$ we need to construct a polynomial-time simulator $\mathcal{S}$ (an ideal adversary) such that for any polynomial-time environment machine $\mathcal{Z}$ and real-world $\mathcal{A}$, it holds that $\mathcal{Z}$ cannot distinguish between the ideal world and the real world with non-negligible advantage. In this sketch we focus on the two cases in which only one of the parties is corrupted. The cases in which both parties are corrupted or both parties are honest are dealt with in a straightforward manner.

The ideal-world adversary, $\mathcal{S}$, begins by setting up an internal simulation of all the real-world parties and functionalities: the sender, the receiver and $\mathcal{F}_{\mathsf{WRAP}}$ (this includes a simulation of the hardware token). Unless explicitly specified by the simulation protocols below, the simulated honest parties and $\mathcal{F}_{\mathsf{WRAP}}$ follow the honest protocol exactly. $\mathcal{S}$ keeps a "simulated view" for each honest party, consisting of the party's input (in the sender's case), its random coins, and the transcript of messages that party received throughout the simulation. At some points in the simulation, $\mathcal{S}$ may "rewrite" the simulated view of an honest party. It makes sure the new simulated view is consistent with any messages previously sent by that party to a corrupt party (note that $\mathcal{F}_{\mathsf{WRAP}}$ can never be corrupted, so messages sent to $\mathcal{F}_{\mathsf{WRAP}}$ may be changed as well).

Corrupted receiver. In the setup phase $\mathcal{S}$ simulates the interaction between $\mathcal{F}_{\mathsf{WRAP}}$, the honest sender and the corrupt receiver. That is, it chooses n random pairs (a_i, b_i) and sends to the simulated copy of $\mathcal{F}_{\mathsf{WRAP}}$ a description of the Turing machine which was specified by the protocol.

Whenever $\mathcal{S}$ receives a message $(\mathsf{receipt}, i)$ from $\mathcal{F}_{\mathsf{BCOM}}$, it chooses a random bit m'_i and simulates the honest sender with input (i, m'_i) interacting with the receiver and with the token. In this case it may be that $\mathcal{S}$ is simulating the honest sender with the wrong input, that is, in the ideal world the sender committed to m_i which is different from m'_i. We claim, however, that the view of the simulated receiver is independent of the committed bit and thus the view of $\mathcal{A}$ is identically distributed in both cases. The view of the simulated receiver consists of $x_i = a_i \oplus b_i \oplus \boldsymbol{m}'_i$ and $y_i \leftarrow a_i \odot c_i \oplus b_i \odot \bar{c}_i$. Each bit of y_i reveals either the corresponding bit of a_i or the corresponding bit of b_i. This implies that the value $a_i \oplus b_i$ is uniformly distributed from the receiver's point of view, and therefore x_i is uniformly distributed as well.

Whenever $\mathcal{S}$ receives a message (open, i, m_i) from $\mathcal{F}_{\mathsf{BCOM}}$ there are two possible cases. In the first case, it holds that m'_i (the bit with which $\mathcal{S}$ simulated the i-th commit stage) is the same as the revealed bit m_i. In this case $\mathcal{S}$ simulates the honest sender following the opening phase of the protocol. The simulation is clearly perfect in this case. In the second case, it holds that $m'_i \neq m_i$. If the simulated receiver sent some c_i to the token in the i-th commit phase, then $\mathcal{S}$ rewrites the history of the simulated sender by replacing a_i and b_i with $\hat{a}_i \leftarrow a_i \oplus \bar{c}_i$ and

$\hat{b}_i \leftarrow b_i \oplus c_i$, and simulates the sender in the opening phase. Note that the new values satisfy $y_i = c_i \odot \hat{a}_i \oplus \bar{c}_i \odot \hat{b}_i$ and $\hat{a}_i \oplus \hat{b}_i \oplus x_i = \boldsymbol{m_i}$ (and are uniformly distributed given this view), and therefore the simulation matches the real world. If the receiver did not send c_i to the token in the i-th commit phase, then $\mathcal{S}$ internally rewinds the token to the point in which $\mathsf{commit}(i, m'_i)$ was invoked and reruns the simulation from that point, replacing m'_i with m_i and leaving all other inputs and random coins without change. Since the token is just part of the simulation of $\mathcal{F}_{\mathsf{WRAP}}$, and $\mathcal{S}$ knows the code it is executing, it can efficiently rewind it. Note that the messages seen by the receiver do not change and therefore the simulation is correct.

Corrupted sender. In the setup phase $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{WRAP}}$ to $\mathcal{A}$ who sends a description of a Turing machine M to $\mathcal{F}_{\mathsf{WRAP}}$. $\mathcal{S}$ now has a description of M and this will be used to rewind the simulated token at a later stage.

Whenever $\mathcal{A}$ initiates a new commit phase (say, the i-th commit phase), $\mathcal{S}$ simulates the honest receiver in this execution. If the commit phase was successful, then $\mathcal{S}$ needs to "extract" the bit m_i to which the corrupted sender committed to in order to instruct the sender in the ideal world to send this bit to $\mathcal{F}_{\mathsf{BCOM}}$. $\mathcal{S}$ rewinds the simulated token to the point in time before the honest receiver sent the challenge c_i to the token. $\mathcal{S}$ instead simulates sending $(i, \bar{c}_i)$ as the challenge. Denote the response y'_i. Now $\mathcal{S}$ "guesses" the values a_i and b_i using the original response y_i and the new response y'_i as follows: if the first bit of c_i was 1, then $\mathcal{S}$ sets the first bit of the guess for a_i to be the first bit of y_i, otherwise it sets the first bit of the guess for b_i to be the first bit of y'_i. Similarly $\mathcal{S}$ "guesses" all the bits of a_i and b_i. Denote by a'_i and b'_i these guesses. $\mathcal{S}$ sets m'_i as the majority of the bit-string $a'_i \oplus b'_i \oplus x_i$. $\mathcal{S}$ then instructs the ideal sender to send $(\mathsf{commit}, i, m'_i)$ to $\mathcal{F}_{\mathsf{BCOM}}$.

Whenever $\mathcal{A}$ initiates a new opening phase by sending (i, a_i, b_i) to the simulated receiver, $\mathcal{S}$ simulates the honest receiver in the opening phase. If $y_i = c_i \odot a_i \oplus \bar{c}_i \odot b_i$ and $x_i \oplus a_i \oplus b_i = \boldsymbol{m}_i$, where m_i is the bit that $\mathcal{S}$ instructed the ideal sender to send to $\mathcal{F}_{\mathsf{BCOM}}$ in the i-th commit phase, then $\mathcal{S}$ instructs the ideal sender to send (open, i) to $\mathcal{F}_{\mathsf{BCOM}}$. In this case we have that $m_i = m'_i$ and therefore the simulation is perfect. If the latter verification step fails, then $\mathcal{S}$ halts. The key point is that this happens only with negligible probability. That is, the probability that the corrupted sender manages to reveal its commitment to a bit different than m'_i is negligible.

In order to prove that the latter probability is indeed negligible, we consider the following game between two provers and a verifier: The verifier chooses a random bit c and sends it to the first prover. The first prover sends a bit y, and the second prover sends the bits (a, b, a', b'). The verifier accepts if $a \oplus b = \neg(a' \oplus b')$ and $y = a \odot c \oplus b \odot \bar{c} = a' \odot c \oplus b' \odot \bar{c}$. If the provers cannot communicate, the verifier will accept with probability at most $1/2$ since for $a \oplus b = \neg(a' \oplus b')$ to hold, either $a \neq a'$ (in which case when $c = 1$ the verifier does not accept) or $b \neq b'$ (in which case when $c = 0$ the verifier does not accept).

In our protocol, if we consider only a single bit from each of the strings a_i, b_i, c_i, x_i and y_i, we can think of the sender and receiver as playing this game:

the receiver is the verifier, and the sender and token are the two provers. The sender "wins" (causes the receiver to accept) if it opens that bit of the commitment to a different value than that "extracted" by the simulator in the commit phase. In the real protocol's commit phase, the game is played k times in parallel. Since the actual bit extracted by the simulator is the *majority* of the bits extracted from each of the games, in order for the sender to successfully open the commitment to a different bit, it must win in at least $k/2$ games. By the Parallel Repetition Theorem [22], the probability that the verifier accepts is exponentially small in the number of parallel repetitions.

4.2 Realizing the Ideal $\mathcal{F}_{\mathsf{MCOM}}$ Functionality

We show a simple variation of protocol $\mathsf{BCOM}_{\mathsf{Goliath}}$ that realizes the ideal $\mathcal{F}_{\mathsf{MCOM}}$ functionality. In the setup phase of $\mathsf{BCOM}_{\mathsf{Goliath}}$, the sender chooses several random pairs (a_i, b_i) and creates a token with these parameters. The number of commitments the protocol supports is therefore limited to the number of such pairs. However, if we are willing to rely on computational assumptions, the pairs (a_i, b_i) can be obtained as the output of a pseudorandom function on input i.

In the setup phase of the new protocol $\mathsf{MCOM}_{\mathsf{Goliath}}$ the sender chooses random seeds a and b for a family of pseudorandom functions $F = \{f_s\}$. The protocol then proceeds exactly as $\mathsf{BCOM}_{\mathsf{Goliath}}$ with the pairs $(f_a(i), f_b(i))$. A formal description of the protocol is provided in Figure 6.

In order to argue that protocol $\mathsf{MCOM}_{\mathsf{Goliath}}$ realizes $\mathcal{F}_{\mathsf{MCOM}}$ we first consider the protocol $\mathsf{BCOM}_{\mathsf{Goliath}}$ when parametrized with $n = 2^k$ (recall that k is the security parameter and n is the number of allowed commitments). With these parameters, the setup phase of the protocol consists of the sender choosing $n = 2^k$ random pairs (a_i, b_i), and the protocol allows the sender and the receiver to perform 2^k commitments – in particular it realizes $\mathcal{F}_{\mathsf{MCOM}}$ in the computational setting (ignoring the fact that the setup phase and the storage required by the token are exponential). Now, we claim that no polynomial-time adversary can distinguish between this protocol and the protocol $\mathsf{MCOM}_{\mathsf{Goliath}}$ with non-negligible probability, as any such adversary can be used in a straightforward manner to distinguish a random function chosen from the family F from a completely random function with non-negligible probability. Therefore, protocol $\mathsf{MCOM}_{\mathsf{Goliath}}$ realizes the ideal $\mathcal{F}_{\mathsf{MCOM}}$ functionality.

Since this proof relies on the seeds of the pseudorandom functions remaining secret from the adversary, it cannot be used to prove security against an adaptive adversary. In particular, the token's response in step 3 of the Commit phase may form a commitment to the seeds (a, b), in which case $\mathsf{MCOM}_{\mathsf{Goliath}}$ is *not* secure against an adaptive adversary.

5 Constructing David Commitments

In this section we describe a protocol that realizes the ideal commitment functionality $\mathcal{F}_{\mathsf{COM}}$ (see Figure 1) without any computational assumptions, where

Protocol $\mathsf{MCOM}_{\mathsf{Goliath}}$

Joint input: a security parameter k, and a family F of pseudorandom functions $f_s : \{0,1\}^k \to \{0,1\}^k$.

Setup phase. The sender chooses random seeds a and b for the family F and creates a token with these parameters. The token also contains state in the form of a counter j initialized to 0. The sender sends the token to the receiver.

Commit phase. Denote the sender's input by (cid, m), and denote by i the number of invocations of the commit phase so far between the sender and the receiver. We assume w.l.o.g. that $cid = i$ (otherwise, both sides can maintain a database that translates between the two).

1. The sender computes $x_i \leftarrow f_a(i) \oplus f_b(i) \oplus \boldsymbol{m}$, and sends (i, x_i) to the receiver.
2. The receiver chooses a random challenge $c_i \overset{R}{\leftarrow} \{0,1\}^k$ and sends (i, c_i) to the token.
3. The token verifies that its internal counter $j = i$ and $j < n$ (otherwise it sends $\perp$ to the receiver and halts). It increments the counter $j \leftarrow j + 1$. The token computes $y_i \leftarrow f_a(i) \odot c_i \oplus f_b(i) \odot \bar{c}_i$ and sends (i, y_i) to the receiver (this is equivalent to letting each bit of c_i choose whether to send the corresponding bit of $f_a(i)$ or the corresponding bit of $f_b(i)$).

Opening phase.

1. The sender sends m and $(i, f_a(i), f_b(i))$ to the receiver.
2. The receiver computes $z_i \leftarrow f_a(i) \oplus f_b(i)$ and verifies that $z_i \oplus x_i = \boldsymbol{m}$ and that $f_b(i) \odot c_i \oplus f_a(i) \odot \bar{c}_i = y_i$. If not, it outputs $\perp$ and halts. Otherwise, it outputs m.

Figure 6: Protocol $\mathsf{MCOM}_{\mathsf{Goliath}}$.

only the receiver creates a hardware token (i.e., the sender is the limited David). In specifying the protocol we again treat the hardware token as one of the protocol participants. The code executed by the token (i.e., the description of the Turing machine M sent to $\mathcal{F}_{\mathsf{WRAP}}$) is implicitly described by the token's role in the protocol.

The intuition behind the protocol is that David can perform a commitment protocol *with the token*. Since there is no communication at all with Goliath, the commitment would be perfectly hiding. In such a case, however, David could postpone the interaction with the token to the opening phase (thus enabling him to equivocate). To overcome this problem, David must prove to Goliath *during the commit phase* that he has already interacted with the token. David does this by giving Goliath a "password" that was contained in the token. However, to prevent the token from using the password to give information about his commitment, David first "tests" Goliath to ensure that he already knows the password. In the opening phase, David sends Goliath a second password (that does depend on the committed bit), which Goliath can verify.

A formal description of the protocol is provided in Figure 7, which is followed by a sketch of its security proof. A complete proof will be provided in the full version of the paper.

Protocol $\mathsf{COM}_{\mathsf{David}}$

Joint input: a security parameter k.

Setup phase. The receiver chooses four random values $s, t \xleftarrow{R} \{0,1\}^k$, $u, v \xleftarrow{R} \{0,1\}^{k/2}$, and creates a token with these parameters. The token contains a flag j initialized to 0. The receiver sends the token to the sender.

Commit phase. Denote the sender's input by $m \in \{0,1\}$.

1. The sender chooses two random values $a, b \xleftarrow{R} \{0,1\}^{k/2}$ and sends a to the token.
2. The token verifies that the internal flag $j = 0$ (otherwise it sends $\perp$ to the sender and halts). It sets $j \leftarrow 1$. The token computes $x \leftarrow u \cdot a + v$ (the computation is in $GF[2^{k/2}]$) and sends s, t and x to the sender.
3. The sender chooses a random challenge $c \xleftarrow{R} \{0,1\}^k$ and sends it the receiver, who replies with $y = s \cdot c + t$ (the computation is in $GF[2^k]$).
4. The sender verifies that $y = s \cdot c + t$, and in this case the sender sends to the receiver s, t, $z = \langle a, b \rangle \oplus m$ and b. If the verification fails then the sender halts.

Opening phase.

1. The sender sends m, a and x to the receiver.
2. The receiver verifies that $x = u \cdot a + v$ and that $z \oplus \langle a, b \rangle = m$. If not, it outputs $\perp$ and halts. Otherwise, it outputs m.

Figure 7: Protocol $\mathsf{COM}_{\mathsf{David}}$.

Proof sketch. We sketch the proof of security for the case of a static adversary. In order to prove that the protocol realizes $\mathcal{F}_{\mathsf{COM}}$ we need to construct simulator $\mathcal{S}$ (an ideal adversary) such that for any polynomial-time environment machine $\mathcal{Z}$ and real-world adversary $\mathcal{A}$, it holds that $\mathcal{Z}$ cannot distinguish between the ideal world and the real world with a non-negligible advantage. We note that our simulator in this proof runs in expected polynomial time (whereas the simulators in the previous section run in strict polynomial time). In this sketch we focus on the two cases in which only one of the parties is corrupted. The cases in which both parties are corrupted or both parties are honest are dealt with in a straightforward manner.

The ideal-world adversary, $\mathcal{S}$, begins by setting up an internal simulation of all the real-world parties and functionalities: the sender, the receiver and $\mathcal{F}_{\mathsf{WRAP}}$. Unless explicitly specified by the simulation protocols below, the simulated honest parties and $\mathcal{F}_{\mathsf{WRAP}}$ follow the honest protocol exactly. $\mathcal{S}$ keeps a "simulated view" for each honest party, consisting of the party's input (in the sender's case), its random coins, and the transcript of messages that party received throughout the simulation. At some points in the simulation, $\mathcal{S}$ may "rewrite" the simulated view of an honest party. It makes sure the new simulated view is consistent with any messages previously sent by that party to a corrupt party (note that $\mathcal{F}_{\mathsf{WRAP}}$ can never be corrupted, so messages sent to $\mathcal{F}_{\mathsf{WRAP}}$ may be changed as well).

Corrupted receiver. In the setup phase $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{WRAP}}$ to $\mathcal{A}$ who sends (on behalf of the simulated receiver) a description of a Turing machine M to $\mathcal{F}_{\mathsf{WRAP}}$. $\mathcal{S}$ now has a description of M and this will be used to rewind the simulated token at a later stage.

When $\mathcal{S}$ receives a message ($\mathsf{receipt}$) from $\mathcal{F}_{\mathsf{COM}}$, it chooses a random bit m' and simulates the honest sender with input m' interacting with the receiver and with the token. In this case it may be that $\mathcal{S}$ is simulating the honest sender with the wrong input, that is, in the ideal world the sender committed to m which is different from m'. We claim however, that the view of the simulated receiver is statistically-close to be independent of the committed bit.

The view of the simulated receiver consists of the challenge c, and of $s, t, z = \langle a, b\rangle \oplus m'$ and b. First, note that if the sender halts before sending the last message of the commit phase, the receiver's view is completely independent of the input bit (since it only affects the last message). So we only need to show that the view is statistically close to independent of m' conditioned on the sender completing the commitment phase successfully.

Since b is only sent in the last message, we can think of it being chosen then. Informally speaking, if there are many values of a for which the token returns some specific s and t, then by choosing b at random, with overwhelming probability $\langle a, b\rangle = 0$ for approximately half of them. Therefore $z = \langle a, b\rangle \oplus m'$ will be almost uniformly distributed, and hence almost independent of m'. If, on the other hand, there are only a few values of a for which the token returns some specific s and t, then the probability that the receiver given a random challenge c can predict $s \cdot c + t$ (and thus allow the sender to complete the commit phase) is negligible.

Whenever $\mathcal{S}$ receives a message (open, m) from $\mathcal{F}_{\mathsf{BCOM}}$ there are two possible cases. In the first case, it holds that m' (the bit with which $\mathcal{S}$ simulated the commit stage) is the same as the revealed bit m. In this case $\mathcal{S}$ simulates the honest sender following the opening phase of the protocol. The simulation is clearly perfect in this case. In the second case, it holds that $m' \neq m$. Denote by a the value that the simulated sender sent to the token in the simulated commit stage. The goal of the simulator is to rewind the simulated token and feed it with random values a' satisfying $\langle a', b \rangle \oplus z = m'$ until either 2^k iterations have passed or until the token outputs (s', t', x') where s' and t' are the same s and t that the token output when it was given a. If more than 2^k iterations have passed, then $\mathcal{S}$ fails and halts. Otherwise, $\mathcal{S}$ simulates the honest sender in the opening phase by sending a', x' and m'. Clearly, if $\mathcal{S}$ does not halt and manages to find such a', then the simulation is correct. In what follows we argue that the expected running time of the simulator is polynomial in the security parameter k.

We show that for any set of random coins of the corrupted receiver, the expected running time of $\mathcal{S}$ is upper bounded by a (fixed) polynomial. Fix the random coins of the receiver, then the receiver and the token define two functions: the token defines the function $T(a) = (s, t, x)$ and the receiver defines the function $y(c)$. Then the expected number of iterations performed by $\mathcal{S}$ is given by

$$\mathrm{E}\left[\mathsf{Iterations}\right] = \sum_{s,t,c} \Pr_a\left[s,t\right] \cdot \Pr_c\left[c\right] \cdot \mathrm{E}\left[\mathsf{Iterations} | s,t,c\right] \quad .$$

Notice that if the tuple (s, t, c) is not consistent, in the sense that $s \cdot c + t \neq y(c)$, then the simulator halts. In addition, conditioned on s and t, the expected number of iterations is independent of c and is equal $1/\Pr_a[s,t]$ (here we ignore the requirement that $\langle a', b \rangle \oplus z = m'$, since b was chosen after s and t were determined, and therefore this requirement will only multiply the expected running time by some constant). Therefore,

$$\begin{aligned} \mathrm{E}\left[\mathsf{Iterations}\right] &= 2^{-k} \cdot \sum_{\substack{\text{consistent } (s,t,c) \\ \text{s.t. } \Pr[s,t]>0}} \Pr_a\left[s,t\right] \cdot \frac{1}{\Pr_a\left[s,t\right]} \\ &= 2^{-k} \cdot \sum_c \left|\{(s,t) : \Pr_a\left[s,t\right] > 0 \text{ and } (s,t,c) \text{ is consistent}\}\right| \quad . \end{aligned}$$

We conclude the argument by showing that the above sum is at most $O(2^k)$ which implies that $\mathrm{E}\left[\mathsf{Iterations}\right]$ is constant. Consider the bipartite graph $G = (L, R, E)$ defined as follows. The left set L is the set of all pairs (s, t) for which $\Pr_a[s,t] > 0$. Notice that since $a \in \{0,1\}^{k/2}$ then $|L| \leq 2^{k/2}$. The right set R is the set of all possible c values, i.e., the set $\{0,1\}^k$. Finally, an edge $((s,t),c)$ exists if the tuple (s, t, c) is consistent, i.e., satisfies $s \cdot c + t = y(c)$. The above sum is exactly the number of edges in the graph: for every $c \in R$ we count the number of incoming edges $((s,t),c)$. The useful property of this graph is that it does not contain any cycles of length 4: it is straightforward to verify that there cannot be two different left-side vertices (s_1, t_1) and (s_2, t_2), and two different right-side vertices c_1 and c_2 that form a cycle of length 4). We can thus use the following theorem to conclude that the number of edges in the graph is at most $O(2^k)$:

Theorem 1 ([3], Chapter 6, Theorem 2.2). *Let $Z(m, n; s, t)$ be the minimal number such that any bipartite graph with vertex parts of orders m and n and $Z(m, n; s, t)$ edges must contain as a subgraph $K_{s,t}$ (the complete bipartite graph with vertex parts of orders s and t). Then*

$$Z(m, n; s, t) < (s-1)^{1/t}(n-t+1)m^{1-1/t} + (t-1)m \ .$$

Note that $K_{2,2}$ is a cycle of length 4, hence the number of edges in the graph is bounded by $Z(2^k, 2^{k/2}; 2, 2) < (2^{k/2} - 1)(2^k)^{1/2} + 2^k < 2^{k+1}$.

Corrupted sender. In the setup phase $\mathcal{S}$ simulates the interaction between $\mathcal{F}_{\mathsf{WRAP}}$, the corrupted sender and the honest receiver. That is, it chooses (on behalf of the receiver) random values s, t, u and v, and sends to the simulated copy of $\mathcal{F}_{\mathsf{WRAP}}$ a description of the Turing machine which was specified by the protocol.

Whenever $\mathcal{A}$ initiates a commit phase, $\mathcal{S}$ simulates the honest receiver in this execution. If the commit phase was successful, then $\mathcal{S}$ needs to "extract" the bit m to which the corrupted sender is committed in order to instruct the sender in the ideal world to send this bit to $\mathcal{F}_{\mathsf{COM}}$. If the corrupted sender sent a value a to the simulated token, then $\mathcal{S}$ computes $m' = \langle a, b\rangle \oplus z$ and instructs the sender in the ideal world to send (commit, m') to $\mathcal{F}_{\mathsf{COM}}$. if the sender did not send any value to the token, $\mathcal{S}$ chooses a random bit m' and instructs the sender in the ideal world to send (commit, m') to $\mathcal{F}_{\mathsf{COM}}$.

Whenever $\mathcal{A}$ initiates a new opening phase by sending (a, x, m) to the simulated receiver, then $\mathcal{S}$ simulates the honest receiver in the opening phase. If $x = u \cdot a + v$ and $z \oplus \langle a, b\rangle = m$, where u, v, z and b are the values from the commit phase, and $m = m'$ where m' is the bit that the ideal sender sent to $\mathcal{F}_{\mathsf{COM}}$ in the commit phase, then $\mathcal{S}$ instructs the ideal sender to send (open) to $\mathcal{F}_{\mathsf{COM}}$. In this case the simulation is perfect. If the latter verification step fails, then $\mathcal{S}$ halts. The key point is that this happens only with negligible probability. That is, the probability that the corrupted sender manages to reveal its commitment to a bit different than m' is negligible. This is because in order to open his commitment to a different bit, the sender must send some $a' \neq a$ in the opening phase. However, to successfully pass verification, the sender must guess the value for $u \cdot a' + v$ (having seen, at most, $u \cdot a + v$). Since these values are independent, the sender guesses correctly with negligible probability.

6 Discussion and Open Problems

Multiple commitment functionality for David. Our protocol for commitment when David is the sender only realizes the $\mathcal{F}_{\mathsf{COM}}$ functionality. Unfortunately, this is an inherent limitation in the protocol rather than an artifact of the proof; when commit is invoked multiple times using the same hardware token, the token's messages when opening a commitment can reveal information about other commitments (that have not yet been opened). Constructing an $\mathcal{F}_{\mathsf{MCOM}}$ functionality for David is an interesting open problem.

We note that the protocol can be composed serially using a single token (as long as every commitment is opened before the next one is invoked). Using the same idea as we did in Goliath's commitment protocol, we can then replace the random values with a pseudorandom function to extend the functionality to any polynomial number of serial invocations (this means that the actual number of hardware tokens needed depends only on the maximum number of concurrent commitments).

Human-compatible commitment for David. David's commitment protocol (cf. Figure 7) requires David to perform a multiplication and an addition operation in a large finite field. While this may be possible to do on paper (or with a calculator), it would be useful to find a protocol that can be performed using simpler operations (as is the case with Goliath's commitment protocols).

Strict polynomial-time simulation for David's commitment. The simulator for David's commitment protocol, unlike Goliath's, runs in expected polynomial time when Goliath is corrupted (rather than strict polynomial time). Although we prove this protocol is secure even for a computationally unbounded Goliath, it is still an interesting question whether a protocol can be constructed with a strict polynomial-time simulator.

Relaxing the physical assumption to tamper-*evident* hardware. Katz's protocol can be implemented using tamper-evident, rather than tamper-proof hardware, if the tokens are returned to their creators after the setup phase. Our bounded commitment protocol can also use this relaxed assumption; since the queries to the token do not depend on the bit to be committed, they can be made ahead of time and the token returned to its owner. This method will not work if the number of commitments is not known ahead of time. Finding a David/Goliath protocol for $\mathcal{F}_{\mathsf{MCOM}}$ based on tamper-evident rather than tamper-proof hardware is an interesting problem.

References

1. B. Barak, R. Canetti, J. B. Nielsen, and R. Pass. Universally composable protocols with relaxed set-up assumptions. In *Proceedings of the 45th Annual Symposium on Foundations of Computer Science*, pages 186–195, 2004.
2. C. H. Bennett and G. Brassard. Quantum cryptography: Public key distribution and coin tossing. In *Proceedings of IEEE International Conference on Computers, Systems and Signal Processing*, pages 175–179, 1984.
3. B. Bollobas. *Extremal Graph Theory.* Courier Dover Publications, 1978.
4. S. Brands. Untraceable off-line cash in wallets with observers. In *Advances in Cryptology - CRYPTO '93*, pages 302–318, 1993.
5. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, 2001. An updated version is available from the Cryptology ePrint Archive, Report 2000/067.

6. R. Canetti, Y. Dodis, R. Pass, and S. Walfish. Universally composable security with global setup. In *Proceedings of the 4th Theory of Cryptography Conference*, pages 61–85, 2007.
7. R. Canetti and M. Fischlin. Universally composable commitments. In *Advances in Cryptology - CRYPTO '01*, pages 19–40, 2001.
8. R. Canetti, E. Kushilevitz, and Y. Lindell. On the limitations of universally composable two-party computation without set-up assumptions. *Journal of Cryptology*, 19(2):135–167, 2006.
9. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally composable two-party and multi-party secure computation. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 494–503, 2002.
10. N. Chandran, V. Goyal, and A. Sahai. Improved UC secure computation using tamper-proof hardware. Cryptology ePrint Archive, Report 2007/334, 2007.
11. D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology - CRYPTO '92*, pages 89–105, 1992.
12. R. Cramer and T. P. Pedersen. Improved privacy in wallets with observers. In *Advances in Cryptology - EUROCRYPT '93*, pages 329–343, 1993.
13. C. Crépeau. Efficient cryptographic protocols based on noisy channels. In *Advances in Cryptology - EUROCRYPT '97*, pages 306–317, 1997.
14. C. Crépeau and J. Kilian. Achieving oblivious transfer using weakened security assumptions. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 42–52, 1988.
15. I. Damgård, S. Fehr, K. Morozov, and L. Salvail. Unfair noisy channels and oblivious transfer. In *Proceedings of the 1st Theory of Cryptography Conference*, pages 355–373, 2004.
16. I. Damgård, J. Kilian, and L. Salvail. On the (im)possibility of basing oblivious transfer and bit commitment on weakened security assumptions. In *Advances in Cryptology - EUROCRYPT '99*, pages 56–73, 1999.
17. I. Damgård, J. B. Nielsen, and D. Wichs. Universally composable multiparty computation with partially isolated parties. Cryptology ePrint Archive, Report 2007/332, 2007.
18. J. Groth and R. Ostrovsky. Cryptography in the multi-string model. In *Advances in Cryptology - CRYPTO '07*, pages 323–341, 2007.
19. D. Hofheinz, J. Müller-Quade, and D. Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *Proceedings of the 5th Central European Conference on Cryptology*, 2005.
20. J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *Advances in Cryptology - EUROCRYPT '07*, pages 115–128, 2007.
21. T. Moran and M. Naor. Basing cryptographic protocols on tamper-evident seals. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, pages 285–297, 2005.
22. R. Raz. A parallel repetition theorem. *SIAM Journal on Computing*, 27(3):763–803, 1998.