BiTR: Built-in Tamper Resilience

Seung Geol Choi[1], Aggelos Kiayias[2], and Tal Malkin[3]

[1] University of Maryland sgchoi@cs.umd.edu
[2] Department of Informatics and Telecommunications, University of Athens aggelos@di.uoa.gr
[3] Columbia University tal@cs.columbia.edu

Abstract. The assumption of the availability of tamper-proof hardware tokens has been used extensively in the design of cryptographic primitives. For example, Katz (Eurocrypt 2007) suggests them as an alternative to other setup assumptions, towards achieving general UC-secure multi-party computation. On the other hand, a lot of recent research has focused on protecting security of various cryptographic primitives against physical attacks such as leakage and tampering.

In this paper we put forward the notion of Built-in Tamper Resilience (BiTR) for cryptographic protocols, capturing the idea that the protocol that is encapsulated in a hardware token preserves its security properties even when an adversary may tamper with its secret state. Our definition is within the UC model, and can be viewed as unifying and extending several prior related works. We provide a composition theorem for BiTR security of protocols, as well as several BiTR constructions for specific cryptographic protocols or tampering function classes. In particular, relaxing the tamper-proof token assumption of Katz's work, we achieve UC-secure computation based on a hardware token that may be susceptible to affine tampering attacks. We also present BiTR proofs for identification and signature schemes in the same tampering model. We next observe that non-malleable codes can be used as state encodings to prove the BiTR property and show new positive results for deterministic non-malleable encodings (as opposed to probabilistic that were previously known) for various classes of tampering functions.

1 Introduction

1.1 Motivation

Security Against Physical Attacks. Traditionally, cryptographic schemes have been analyzed assuming that an adversary has only *black-box* access to them. For example, traditional security definitions for encryption schemes address an adversary who is given the public key — but not the private key — and tries to guess something about the plaintext of a challenge ciphertext, by applying some black-box attack (CPA, CCA, etc.) In practical situations, however, an adversary can often do more. For example, practical hardware devices leak information via numerous side channels, including power consumption [KJJ99], electromagnetic radiation [QS01], and timing [BB05]. When an adversary takes hold of the computational device (e.g., smart-cards or mobile phones), such side channel information is readily accessible. Starting with the work of [ISW03,MR04] there has been a surge of research activity on leakage-resilient cryptographic schemes (cf., [SMY09,AGV09,DKL09,P09,NS09,ADW09,KV09,FKPR10,DGK+10,FRR+10,BG10,DP10,JV10,GR10]).

The present work addresses *tampering attacks*, where an adversary can modify the secret data by applying various physical attacks (c.f., [BDL01,BS97]). Currently, there are only a few results in this area [GLM+04,IPSW06,DPW10]. We focus on hardware tokens that implement two party cryptographic protocols, which can be used directly (e.g., for identification), or as part of other (UC-secure) protocols.

Hardware Tokens. As discussed above, cryptographic primitives have traditionally been assumed to be tamper (and leakage) proof. In the context of larger cryptographic protocols, there have been many works

that (implicitly or explicitly) used secure hardware as a tool to achieve security goals that could not be achieved otherwise. The work most relevant to ours is that of Katz [K07], who suggests to use *tamper-proof hardware tokens* to achieve UC-secure [C01] commitments. This allows to achieve general feasibility results for UC-secure well-formed multi-party computation, where the parties, without any other setup assumptions, send each other tamper-proof hardware tokens implementing specific two-party protocols. There were several follow-up works such as [MS08,CGS08,DNW08,GIS+10,K10,GIMS10], all of which assume a token that is tamper proof.

Given the wide applicability of tamper-proof tokens on one hand, and the reality of tampering attacks on the other, we ask the following natural question:

Can we relax the tamper-proof assumption, and get security using tamperable *hardware tokens?*

Clearly, for the most general interpretation of this question, the answer is typically negative. For example, if the result of [K07] was achievable with arbitrarily-tamperable hardware token, that would give general UC-secure protocols in the "plain" model, which is known to be impossible [CF01]. In this work we address the above question in settings where the class of possible tampering functions and/or the class of protocols we wish to put on a token and protect are restricted.

1.2 Our Contributions

BiTR Definition. We provide a definition of Built-in Tamper Resilience (BiTR) for two party cryptographic protocols, capturing the idea that the protocol can be encapsulated in a hardware token, whose state may be tamperable. Our definition is very general, compatible with the UC setting [C01], and implies that any BiTR protocol can be used as a hardware token within larger UC-protocols. Our definition may be viewed as unifying and generalizing previous definitions [GLM+04,IPSW06,DPW10] (see Section 1.3), and bringing them to the UC setting.

BiTR is a property of a cryptographic protocol M. In short, BiTR can be summarized as follows: any adversary that is able to tamper tokens running M can be simulated by an adversary that has no tampering capability, independently of the environment the tokens may be deployed. The definition has some tunable parameters:

- *A Class of Tampering Functions* $\mathcal{T}$. The bigger the class of tampering functions that we provide resilience against, the better.
- *A State Encoding* ψ. The higher the rate of the encoding the better. Moreover, the less fresh randomness necessary to be generated by the protocol, the better.

While the result one would ideally want — arbitrary BiTR protocols against arbitrary tampering functions, without any expanding encoding on the state — is not achievable, we provide several specific results that trade off these parameters (see below), as well as the following general theorem.

BiTR Composition. As BiTR is a protocol centric property, the natural question that arises is whether it is preserved under composition. A useful result for a general theory of BiTR cryptography would be a general composition theorem which allows combining a BiTR protocol calling a subroutine and a BiTR implementation of that subroutine into one overall BiTR protocol.

Unfortunately we provide specific counter examples showing that a very general composition theorem is not possible. We also point out difficulties in achieving (and even defining) such a general composition theorem, as it is not obvious what should be the right interface to give to an adversary tampering with a subroutine call that is in a hardware token.

More importantly, we formulate and prove a limited composition theorem, which allows composition of a certain type of BiTR protocols. Our main positive construction will use this composition theorem with a simple component, and we hope that future work can take advantage of the full power of our composition theorem for further positive results.

BiTR Constructions without State Encoding. We describe results that require no encoding as compared to their no-tampering-allowed counterparts. It may come as a surprise that it is possible to prove a cryptographic protocol BiTR without any encoding and thus without any validation of the secret protocol state whatsoever. This stems from the power of our definitional framework for BiTR and the fact that it is achieved for specially selected and designed protocols and classes of tampering functions. We define the class $\mathcal{T}_{\mathsf{aff}} = \{f_{a,b} \mid a \in \mathbb{Z}_q^+, b \in \mathbb{Z}_q, f_{a,b}(v) := av + b \bmod q\}$. That is, the adversary may apply a modular affine function of his choice to tamper the state. Affine tampering is an interesting class to consider as it has as special cases multiplication (e.g., shifting — which may be the result of tampering shift-register based memory storage), or addition (which may be result of bit flipping tampering).

We prove three protocols BiTR with respect to this class (without any modification or encoding necessary). The first one is Schnorr's identification (3-move) protocol [S91]. The second is Okamoto's signature scheme [O06]. Both protocols are interesting on their own (e.g., previous work [GLM+04] focused mostly on signature schemes), but the latter is also useful for the third protocol we prove affine-BiTR, described next.

UC-Secure Computation from tamperable tokens. We provide a generalization of Katz's approach [K07] for building UC-secure computation using hardware tokens; this involves the introduction of a commitment scheme with a special property, called a *dual-mode parameter generation (DPG)* — depending on the mode of the parameter, the commitment scheme is either statistically hiding or a trapdoor commitment. We then observe that any commitment endowed with a DPG is sufficient for providing UC-secure multi-party computation assuming tamper proof tokens. Following this track, we present a new DPG protocol for a commitment scheme, BiTR against affine tampering functions, that relies on discrete-log based primitives including the digital signature scheme of Okamoto [O06]. Thus, we obtain UC-secure general computation using hardware tokens tamperable with affine functions. We then examine a different class of tokens that implement a single OT [GKR08] and were utilized in [GIS+10] for UC secure computation. We characterize the functions against which these tokens are BiTR.

BiTR Constructions with State Encoding. We next discuss how one can take advantage of state consistency checks to design BiTR protocols. We observe first that non-malleable codes [DPW10] can be used as an encoding for proving the BiTR property of protocols. This gives rise to the problem of constructing such codes. Existing constructions [DPW10] utilize randomness in calculating the encoding; we provide new constructions for such encodings focusing on purely *deterministic constructions*. In fact, when the protocol uses no randomness (e.g., a deterministic signing algorithm) or a finite amount of randomness (e.g., a prover in the resettable zero-knowledge [CGGM00] setting), by using deterministic encodings the token may dispense with the need of random number generation.

Our design approach takes advantage of a generalization of non-malleable encodings (called δ-non-malleable), and we show how they can be constructible for any given set of tampering functions (as long as they exist). While the construction is not very efficient, by assuming that each function in the class $\mathcal{T}$ works independently on small blocks (of logarithmic size) we provide a construction of efficient deterministic non-malleable codes by concatenating Reed-Solomon codes with a collection of specific instances of non-malleable codes (that apply in each coordinate independently).

1.3 Related Work

We briefly describe the most relevant previous works addressing protection against tampering.

Gennaro et al. [GLM+04] considered a device with two separate components: one is tamper-proof yet readable (circuitry), and the other is tamperable yet read-proof (memory). They defined algorithmic tamper-proof (ATP) security and explored it's possibility for signature and decryption devices. To overcome arbitrary effective tampering attacks, they had to introduce self-destruct and public-key parameters, with which they were able to demonstrate ATP constructions. Their definition of ATP security was given only for the specific tasks of signature and encryption. In contrast, our definition is simulation based, independent of the correctness or security objectives of the protocol, and we consider general two-party protocols (and the implications in the UC framework [C01,K07]).

Ishai et al. [IPSW06] considered an adversary who can tamper with the wires of a circuit. They showed a general compiler that outputs a self-destructing circuit that withstands such a tampering adversary. Considering that memory corresponds to a subset of the wires associated with the state in their model, the model seems stronger than ours (as we consider only the state, not the computation circuit). However, the tampering attack they considered is very limited: it modifies a bounded subset of the wires between each invocation, which corresponds to tampering memory only partially.

Dziembowski et al. [DPW10] introduced the notion of non-malleable codes and tamper-simulatability to address similar concerns as the present work. A distinguishing feature from the approach of that work is that BiTR is a protocol centric property. As such, it allows arguing about tamper resilience while taking advantage of specific protocol design features that enable BiTR even without any encodings. This can be advantageous as the introduction of additional circuitry or a randomness device (as required by the positive results of [DPW10]) may be infeasible or uneconomical — or even unsafe in practice as it may introduce new pathways for other attacks. In contrast, our positive results do not require state encodings or when they do, they do not rely on randomness and thus they can be used to fortify protocols into BiTR even in the case when a randomness device is not accessible (or reliable) by the token.

Another important difference between our work and all the ones above, is that we address the universally composable setting, rather than a stand-alone notion and thus the BiTR property would be preserved in all deployment environments.

2 BiTR Definitions

Ideal functionalities $\mathcal{F}_{wrap}$ and $\mathcal{F}_{twrap}$. Katz [K07] modeled usage of a tamper-proof hardware token as an ideal functionality $\mathcal{F}_{wrap}$ in the UC framework. Here, we slightly modify the functionality so that it is

$\mathcal{F}_{wrap}(M)$ is parameterized by a polynomial p and a security parameter k. $\mathcal{F}_{wrap}$ proceeds as follows:

Create: Upon receiving $\langle Create, sid, P, P', msg\rangle$ from party P:

1. Let $msg' = (Initialize, msg)$. Run $M(msg')$ for at most $p(k)$ steps.
2. Let out be the response of M (set out to $\perp$ if M does not respond). Let s' be the updated state of M.
3. Send $\langle Initialized, sid, P', out\rangle$ to P and $\langle Create, sid, P, P'\rangle$ to P'.
4. If there is no record $(P, P', *, *)$, then store (P, P', M, s').

Forge: Upon receiving $\langle Forge, sid, P, P', M', s\rangle$ from the adversary, if P is not corrupted, do nothing. Otherwise do:

1. Send $\langle Create, sid, P, P'\rangle$ to P'.
2. If there is no record $(P, P', *, *)$, then store (P, P', M', s).

Run: Upon receiving $\langle Run, sid, P, msg\rangle$ from party P', find a record (P, P', K, s). If there is no such record, do nothing. Otherwise, do:

1. Run $K(msg; s)$ for at most $p(k)$ steps.
2. Let out be the response of K (set out to $\perp$ if K does not respond). Let s' be the updated state of K. Send (sid, P, out) to P'.
3. Update the record with (P, P', K, s').

$\mathcal{F}_{twrap}(M, \mathcal{T}, \psi)$, also parameterized by p and k (and $\psi = (E, D)$ is an encoding scheme), proceeds as follows

Create: As in $\mathcal{F}_{wrap}(M)$ with the only change that state s' is stored as $E(s')$ in memory.

Forge: As in $\mathcal{F}_{wrap}(M)$.

Run: Upon receiving $\langle Run, sid, P, msg\rangle$ from party P', find a record $(P, P', K, \tilde{s})$. If there is no such record, do nothing. Otherwise, do:

1. (Decoding) If P is corrupted, set $s = \tilde{s}$; otherwise, set $s = D(\tilde{s})$. If $s = \perp$, send $(sid, P, \perp)$ to P' and stop.
2. Run $K(msg; s)$ for at most $p(k)$ steps.
3. Let out be the response of K (set out to $\perp$ if K does not respond). Let s' be the updated state of K. Send (sid, P, out) to P'.
4. (Encoding) If P is corrupted, set $\tilde{s} = s'$; otherwise set $\tilde{s} = E(s')$.
5. Update the record with $(P, P', K, \tilde{s})$.

TamperRun: Upon receiving $\langle TamperRun, sid, P, P', \tau, msg\rangle$ from the adversary $\mathcal{A}$, if P' is not corrupted, do nothing. Otherwise; find a record $(P, P', K, \tilde{s})$. If there is no such record, do nothing. Otherwise, do:

1. Set $\tilde{s} = \tau(\tilde{s})$. Follow steps $1 \sim 5$ in handling **Run** request.

Fig. 1. Ideal functionalities $\mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap}(M, \mathcal{T}, \psi)$

parameterized by an interactive machine (ITM) M for a two-party protocol[4] (see Fig. 1). The modification does not change the essence of the wrapper functionality; it merely binds honest parties to the use of a specific embedded program. Corrupted parties may embed an arbitrary program in the token by invoking $Forge$. Given that we will be interested in composition between tokens we will also consider $\mathcal{F}_{wrap}(M; K)$ as the compound functionality of a wrapped protocol M that has access to an oracle $\mathcal{F}_{wrap}(K)$.

We also define a new functionality $\mathcal{F}_{twrap}$ similar to $\mathcal{F}_{wrap}$ but with tampering allowed. Let $\mathcal{T}$ be a collection of (randomized) functions. Let $\psi = (E, D)$ be an encoding scheme[5]. The essential difference between $\mathcal{F}_{twrap}$ and $\mathcal{F}_{wrap}$ is the ability of the adversary to tamper with the internal state of the hardware token by executing tampered runs — a function drawn from $\mathcal{T}$ is applied on the internal state of the hardware token. This (weaker) ideal functionality notion is fundamental for the definition of BiTR that comes next.

BiTR Protocols. We define a security notion for a protocol M, called Built-in Tamper Resilience (BiTR), which essentially requires that $\mathcal{F}_{twrap}(M)$ is interchangeable with $\mathcal{F}_{wrap}(M)$. We adopt the notations in the UC framework given by Canetti [C01].

[4] We will interchangeably use protocols and ITMs.

[5] We will sometimes omit ψ from $\mathcal{F}_{twrap}$ when it is obvious from the context.

Definition 1 (BiTR protocol). *The protocol M is $(\mathcal{T}, \psi)$-*BiTR *if for any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that for any non-uniform PPT $\mathcal{Z}$,*

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}} ,$$

where $\approx$ denotes computational indistinguishability.

In case $\psi = \mathsf{id}$ we simply write $\mathcal{T}$*-BiTR*. Note that this definition is given through the ideal model, which implies (by the standard UC theorem) that whenever a tamper-proof token wrapping M can be used, it can be replaced by a $\mathcal{T}$-tamperable token wrapping M.[6] As a trivial example, every protocol is $\{\mathsf{id}\}$-BiTR, where id is the identity function.

3 Composition of BiTR Protocols

Consider an ITM K and another ITM M that calls K as a subroutine. We denote by $(M; K)$ the compound ITM. The internal state of $(M; K)$ is represented as the concatenation of the two states $s_M||s_K$ where s_M is a state of M and s_K the state of K at a certain moment of runtime.

Let $\mathcal{F}_{twrap}(M; K, \mathcal{T}_1 \times \mathcal{T}_2, \psi_1 \times \psi_2)$ denote an ideal functionality that permits tampering with functions from $\mathcal{T}_1$ for the state of M and from $\mathcal{T}_2$ for the state of K while the states are encoded with ψ_1 and ψ_2 respectively. This extends the definition of $\mathcal{F}_{twrap}$. Note that one $TamperRun$ of $(M; K)$ may call K many times. In this case, tampering on K is applied only once at the beginning of the $TamperRun$ of $(M; K)$. Using this notation we recast the notion of BiTR for protocols that may call subroutines[7].

Definition 2 ("Hybrid" BiTR protocol). *The protocol M with subprotocol K is $(\mathcal{T}_1, \psi_1)$-*BiTR *if for any PPT $\mathcal{A}$ and any $\mathcal{T}_2, \psi_2$, there exists a PPT $\mathcal{S}$ such that for any non-uniform PPT $\mathcal{Z}$,*

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\mathcal{T}_1\times\mathcal{T}_2,\psi_1\times\psi_2),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\{\mathsf{id}\}\times\mathcal{T}_2,\mathsf{id}\times\psi_2),\mathcal{S},\mathcal{Z}},$$

The above definition enables us to consider whether a certain protocol is BiTR by itself without considering the properties of any subprotocols it may call. Armed with the above definition we can now return to the question of composition: *Given protocol M with subprotocol K that are both BiTR for tampering classes $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively, is the compound protocol (i.e., $M; K$) BiTR against the tampering class $\mathcal{T}_1 \times \mathcal{T}_2$?* Unfortunately, the answer in general is no. To see this consider the following examples.

BiTR Composition Counter-Example #1. Suppose there are two ITMs M and K where M is $\{\mathsf{id}\}$-BiTR and K is a signature scheme $\mathcal{T}$-BiTR against some $\mathcal{T}$. M operates as follows: Given an encryption key pk as input, pick a random number r, call K for a signature on r, and output $E_{pk}(r, \mathsf{Sig}(r))$ where E is a non-malleable encryption scheme. Now, some tampering function (id, τ) with $\tau \in \mathcal{T}$ is applied to $(M; K)$. Any tampered execution in the $\mathcal{F}_{twrap}(\cdot)$ setting will produce some ciphertext $C' = E_{pk}(r, \sigma')$ where σ' is a signature that reflects the tampering application of τ. For the resulting protocol to be BiTR, the simulator has to construct such C' from the results of non-tampered executions (i.e., ciphertexts $C = E_{pk}(r, \sigma)$ that come from $\mathcal{F}_{wrap}(M; K)$, where σ is a valid signature that has not been tampered with). However, once

[6] One could also consider a definition that requires this in the context of a *specific* UC-protocol. We believe our stronger definition, which holds for *any* UC-protocol using a token with M, is the right definition for built-in tamper resilience.

[7] We will use subroutines and subprotocols interchangeably.

the simulator generate such C', it will violate the non-malleability of E_{pk}, since it amounts to generating a related ciphertext C' from a ciphertext C. In this scenario, the BiTR simulator for K is of no use; it is useful only if direct access to r and σ — rather than its encryption — is allowed.

As the above counter-example indicates, whether the compound protocol of two BiTR protocols is BiTR depends on how the outer protocol M uses the subprotocol K. An intuition is that the ITM M should preserve the output from K (say, o_K) in its own output in a malleable form; this may enable the simulator of $\mathcal{F}_{wrap}(M; K)$ to modify o_K appropriately. To capture this requirement, we will introduce the condition of a *transparent subprotocol* below. Roughly speaking it requires that the modifications needed due to tampering of the subprotocol K have to be "transparent" to the external output of $(M; K)$.

Still, even if M is calling K in an entirely transparent fashion (e.g., simply passing through the output of K for some public input) the composition of two BiTR protocols might still fail, as demonstrated next.

BiTR Composition Counter-Example #2. Suppose K implements a signature algorithm with state $b||sk$ where $b \in \{0, 1\}$ and sk is a signing key. K operates as follows : given an input m it reads b and if $b = 0$ it returns a signature on m. On the other hand, if $b = 1$ it returns two signatures on the messages $m||0$ and $m||1$. Normally it is always the case that $b = 0$. Consider now a class of tampering functions $\mathcal{T}$ that contains a single function t which switches the first bit of the internal state from 0 to 1. Observe that it is easy to show that K is BiTR for $\mathcal{T}$ — indeed we can build a simulator that accesses the protocol K twice for the messages $m||0$, $m||1$ and returns the two signatures. Now consider a protocol M that calls K as follows: on input a length 1^k, M selects a random string r of length k and calls K; M will return the response of K whatever this might be. Now suppose we would like to show that $(M; K)$ is $(\{\mathsf{id}\} \times \mathcal{T})$-BiTR. It can be easily seen that any simulator is bound to fail due to the unforgeability of the signature: once tampering of the inner subprotocol occurs the simulator will have to present two signatures on the messages $r||0$ and $r||1$; while the simulator is allowed to call $\mathcal{F}_{wrap}(M; K)$ as many times as it wants, the fact that M randomizes the calls to K make it infeasible to recover two signatures on two messages that differ only on 1 bit.

It follows that BiTR composition would be sensitive to the resources that the simulator of K needs to utilize in order to simulate the tampering operation. To address these issues we introduce the notion of BiTR-parsimonious protocols below.

The BiTR Composition Theorem. Before expressing the requirements for our composition theorem that were motivated by the above counterexamples, we need to establish some notation. We first consider how we can extend an environment $\mathcal{Z}$ that operates with $(M; K)$ tokens to an environment that operates with K tokens through the interface of M. While this extension is mostly straightforward, some care needs to be applied when considering the interaction with the adversary.

- For a non-uniform PPT environment $\mathcal{Z}$ that operates with $(M; K)$ tokens, we define the *layered environment* $\mathcal{Z}^M$ of $\mathcal{Z}$ (with the M-layer) that operates with K tokens. Roughly speaking, it simulates the behavior of the M-layer for all $(M; K)$ invocations. It works as follows:
  1. Communication between $\mathcal{Z}$ and the adversary is simply passed without any alteration.
  2. Upon receiving a $Create$ command from $\mathcal{Z}$, $\mathcal{Z}^M$ correspondingly simulates an instance of M. If the token M issues a $Create$ instruction to a K token then $\mathcal{Z}^M$ makes the corresponding call (to $\mathcal{F}_{wrap}(K)$ or $\mathcal{F}_{twrap}(K, \cdot)$). Upon receiving a request for forging a token M by the adversary, $\mathcal{Z}^M$ will comply and generate an instance of this token with the program supplied by the adversary; any $Forge$ commands for K tokens will be sent back to the adversary. $\mathcal{Z}^M$ will simulate the all

responses at the end of token creation for $\mathcal{Z}$, including the $Initialized$ command back to calling party P and the $\langle Create, sid, P, P' \rangle$ to the prospective token user P'.

3. Upon receiving a Run command from $\mathcal{Z}$, $\mathcal{Z}^M$ runs the corresponding instance of M (or the forged algorithm in case of a forged program). If M issues a Run instruction to a K token then $\mathcal{Z}^M$ passes it on. In the end, $\mathcal{Z}^M$ will return the outcome of the computation of the token back to $\mathcal{Z}$.
   If the adversary asks to $Run/TamperRun$ a token M, $\mathcal{Z}^M$ will comply and run the M token but it will not permit tampering. If the instance of M that the adversary invoked wishes to make some invocation to a K token, then $\mathcal{Z}^M$ will pass this back to the adversary, informing it that the invocation it requested wishes to call K and will hold the execution of M for when the answer arrives.
4. In the end $\mathcal{Z}^M$ will output whatever $\mathcal{Z}$ outputs.

– For a PPT adversary $\mathcal{A}$ against $(M; K)$ tokens and an ITM I, we define the corresponding layered adversary $\mathcal{A}^{M;I}$ (with the M-layer and interface I) that attacks K tokens. It works exactly in the same way as $\mathcal{A}$ except for the handling of the $(M; K)$ tokens. Whenever $\mathcal{A}$ invokes an $(M; K)$ token, $\mathcal{A}^{M;I}$ passes this call to the environment instead (this would hold true for tamper and non-tamper runs). If the environment requests a (possible tamper) run to K while handling the invocation of the M token the following will take place: $\mathcal{A}^{M;I}$ will pass this input to I to get it possibly modified and then proceed to invoke K (to $\mathcal{F}_{wrap}(K)$ or $\mathcal{F}_{twrap}(K, \cdot)$) with the possibly modified message; when the response from K arrives $\mathcal{A}^{M;I}$ will pass it to I again for possible post-processing and then return the result to the environment. In the end $\mathcal{A}^{M;I}$ will obtain the output of the invocation of the $(M; K)$ token from the environment and return it to $\mathcal{A}$ who will receive it as the response of the (possibly tamper) run of the $(M; K)$ token.
  The term layered (with the M-layer) comes from the fact that $\mathcal{A}^{M;I}$ passes to the environment requests to invoke $(M; K)$ tokens. The most important restriction of $\mathcal{A}^{M;I}$ compared to a general attacker against K tokens is that the layered adversary lets the environment drive the attack against K tokens following the way the M-layer of a hardware token uses K.

We next define a strengthening of the BiTR property for protocols K that can be subprotocols of other protocols M. We use the notion of layered adversary $\mathcal{A}^{M;I}$; for simplicity when I is "pass-through", i.e., it implements the identity function we write $\mathcal{A}^M$, omitting I from the superscript.

Definition 3 (BiTR-parsimonious). *We say the ITM K is $(\mathcal{T}, \psi)$-BiTR parsimonious if for any M that calls K as a subprotocol the following holds: for all adversaries $\mathcal{A}$ against $(M; K)$ there is $\mathcal{A}', I$ such that for all environments $\mathcal{Z}$ it holds that :*

$$\text{IDEAL}_{\mathcal{F}_{twrap}(K,\mathcal{T},\psi),\mathcal{A}^M,\mathcal{Z}^M} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(K),(\mathcal{A}')^{M;I},\mathcal{Z}^M}.$$

The critical difference between BiTR and BiTR parsimonious is in the nature of the adversary/simulator: in particular $(\mathcal{A}')^{M;I}$ is only allowed to access $\mathcal{F}_{wrap}(K)$ when it is being asked in the context of a calling procedure M and do that only as many times as M dictates. It is easy to see that BiTR parsimonious implies BiTR by simply letting the calling procedure M be just a pass-through function as well as the interface I. In such case, the simulator $(\mathcal{A}')^{M;I}$ can in fact be used for any environment $\mathcal{Z}$ against K tokens. Much of the difficulty in proving the BiTR parsimonious property is the construction of the interface I; to reflect this we may call a protocol BiTR *parsimonious with interface I*.

Finally we introduce the transparent subprotocol property for a pair of protocols M and K where the first calls the second as a subprotocol. The transparency condition effectively states that in any environment

where $(M;K)$ tokens are used, given an adversary that runs forged tokens of K driven by the way M is using the K tokens, we can derive a simulator that handles only $(M;K)$ tokens.

Definition 4 (Transparent Subprotocol). *Consider a protocol M with subprotocol K. K is a transparent subprotocol of M with respect to I, if for any $\mathcal{A}$, there is an $\mathcal{A}'$ such that for all non-uniform $\mathcal{Z}$ it holds that*

$$\text{IDEAL}_{\mathcal{F}_{wrap}(K),\mathcal{A}^{M;I},\mathcal{Z}^M} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M;K),\mathcal{A}',\mathcal{Z}}.$$

It is worth noting that the major issue with proving transparency is that a simulator should be built that can translate the malicious behavior of any adversary against K tokens while just controlling $(M;K)$ tokens "from the outside."

We are now ready to state our composition theorem. Due to space limitations, all proofs appear in the appendix.

Theorem 1 (BiTR Composition Theorem). *Let M be a protocol and K be a subprotocol of M. If M is $(\mathcal{T}_1,\psi_1)$-BiTR, K is $(\mathcal{T}_2,\psi_2)$-BiTR parsimonious with interface I and K is a transparent subprotocol of M with respect to I, then $(M;K)$ is $(\mathcal{T},\psi)$-BiTR where $\mathcal{T}=\mathcal{T}_1\times\mathcal{T}_2$, and $\psi=\psi_1\times\psi_2$.*

Proof. Consider some $\mathcal{A}$. We will show that there is some $\mathcal{S}$ for which it holds that for any $\mathcal{Z}$

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\mathcal{T}_1\times\mathcal{T}_2,\psi_1\times\psi_2),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M;K),\mathcal{S},\mathcal{Z}} \tag{1}$$

For simplicity we will assume that $\mathcal{Z}$ runs only a single instance of an $(M;K)$ token; a proof of the general case can be easily derived by employing a standard hybrid argument.

Using the BiTR property of M we know that there is an $\mathcal{S}_1$ for which it holds that for any $\mathcal{Z}$,

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\mathcal{T}_1\times\mathcal{T}_2,\psi_1\times\psi_2),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\{id\}\times\mathcal{T}_2,id\times\psi_2),\mathcal{S}_1,\mathcal{Z}} \tag{2}$$

Note that $\mathcal{S}_1$ is an adversary that may tamper with the inner token only. Specifically it executes tampered runs for $M;K$ tokens for which it can specify a tampering for the state of the K protocol.

Now recall the definition of the M-layer adversaries and environments. By considering the M-layer of $\mathcal{S}_1$ it is immediate that for any $\mathcal{Z}$, we have :

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\{id\}\times\mathcal{T}_2,id\times\psi_2),\mathcal{S}_1,\mathcal{Z}} = \text{IDEAL}_{\mathcal{F}_{twrap}(K,\mathcal{T}_2,\psi_2),\mathcal{S}_1^M,\mathcal{Z}^M} \tag{3}$$

Now due to the fact that K is BiTR-parsimonious we have that there is an $\mathcal{S}_2$ that depends only on $\mathcal{S}_1^M$ for which it holds that for any environment $\mathcal{Z}$ that operates with $(M;K)$ tokens we have for some interface I

$$\text{IDEAL}_{\mathcal{F}_{twrap}(K,\mathcal{T}_2,\psi_2),\mathcal{S}_1^M,\mathcal{Z}^M} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(K),\mathcal{S}_2^{M;I},\mathcal{Z}^M} \tag{4}$$

Now using the transparent subprotocol property w.r.t. I we have that there is an $\mathcal{S}_3$ such that for any $\mathcal{Z}$,

$$\text{IDEAL}_{\mathcal{F}_{wrap}(K),\mathcal{S}_2^{M;I},\mathcal{Z}^M} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M;K),\mathcal{S}_3,\mathcal{Z}} \tag{5}$$

Now we return to the original statement (1) we need to show. We set $\mathcal{S} = \mathcal{S}_3$. Consider any $\mathcal{Z}$.

$$\begin{aligned}
&\text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\mathcal{T}_1\times\mathcal{T}_2,\psi_1\times\psi_2),\mathcal{A},\mathcal{Z}} \\
&\overset{(2)}{\approx} \text{IDEAL}_{\mathcal{F}_{twrap}(M;K,\{id\}\times\mathcal{T}_2,id\times\psi_2),\mathcal{S}_1,\mathcal{Z}} \\
&\overset{(3)}{=} \text{IDEAL}_{\mathcal{F}_{twrap}(K,\mathcal{T}_2,\psi_2),\mathcal{S}_1^M,\mathcal{Z}^M} \\
&\overset{(4)}{\approx} \text{IDEAL}_{\mathcal{F}_{wrap}(K),\mathcal{S}_2^{M;I},\mathcal{Z}^M} \\
&\overset{(5)}{\approx} \text{IDEAL}_{\mathcal{F}_{wrap}(M;K),\mathcal{S}_3,\mathcal{Z}}
\end{aligned}$$

This completes the proof. ■

4 Affine BiTR Protocols without State Encoding

In this section, we show two protocols (for identification and signatures, respectively) that are BiTR against certain tampering functions, without using any encoding scheme. Specifically, we consider a tampering adversary that can modify the state of the hardware with *affine functions*. Assuming the variables in the hardware are elements of $\mathbb{Z}_q$, the adversary can choose a tampering $f_{a,b}$ on a variable v, which will change v into $f_{a,b}(v) = av + b \bmod q$. Let $\mathcal{T}_{\mathsf{aff}} = \{f_{a,b} \mid a \in \mathbb{Z}_q^+, b \in \mathbb{Z}_q\}$.

Schnorr Identification [S91**].** The Schnorr identification is a three-move two party protocol between a prover and a verifier. The common input is $y = g^x$, where g is a generator of a cyclic group of size q, and the prover's auxiliary input is $x \in \mathbb{Z}_q$. The protocol proceeds as follows:

1. The prover picks a random $t \in \mathbb{Z}_q$ and sends $z = g^t$ to the verifier.
2. The verifier picks a random $c \in \mathbb{Z}_q$ and sends c to the prover.
3. The prover computes $s = cx + t \bmod q$ to the verifier, who checks if $zy^c = g^s$.

We consider an ITM on the prover side wrapped as a hardware token. The description is found in Fig. 2.

Theorem 2. *The ITM M_{sch} in Fig. 2 is $\mathcal{T}_{\mathsf{aff}}^2$-BiTR without any encoding.*

Proof. Let M be an ITM as described above and $\mathcal{T} = \mathcal{T}_{\mathsf{aff}}^2$. We show that for any non-uniform PPT $\mathcal{Z}$ and any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}}.$$

Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted.

Fix $\mathcal{Z}$ and $\mathcal{A}$. For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(M, \mathcal{T})$. In order to keep the history of tamperings, $\mathcal{S}$ maintains two functions f_x and f_t, which are initialized with identity functions. The simulator $\mathcal{S}$ proceeds as follows:

– $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.

- Upon receiving $\langle Create, sid, P', P, msg\rangle$ from $\mathcal{A}$ on behalf of P':
  $\mathcal{S}$ forwards the message to $\mathcal{F}_{wrap}$ and sends its response to $\mathcal{A}$ in return.
- Upon receiving $\langle Forge, sid, P', P, M', s'\rangle$ from $\mathcal{A}$ on behalf of P':
  $\mathcal{S}$ forwards the message to $\mathcal{F}_{wrap}$.
- Upon receiving $\langle Run, sid, P, msg\rangle$ from $\mathcal{A}$ on behalf of P':
  If $msg = (Prove, \perp)$, $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle Run, sid, P, msg\rangle$. Let $\langle sid, P, z\rangle$ be the output from $\mathcal{F}_{wrap}$. $\mathcal{S}$ forwards the output $\mathcal{A}$ and sets f_t to the identity function.

  If $msg = (Prove, c')$, let w and v (resp., a and b) be the cofficients of f_x (resp., f_t), that is, $f_x(z) = wz + v$ and $f_t(z) = az + b$. $\mathcal{S}$ computes $c = wc'/a$ and calls $\mathcal{F}_{wrap}$ with $\langle Run, sid, P, (Prove, c)\rangle$. Let $\langle sid, P, s\rangle$ be the output from $\mathcal{F}_{wrap}$. $\mathcal{S}$ computes $s' = as + c'v + b$ and sends $\langle sid, P, s'\rangle$ to $\mathcal{A}$. Set f_t to the identity function.
- Upon receiving $\langle TamperRun, sid, P, P', (f_{a_x,b_x}, f_{a_t,b_t}), msg\rangle$ from $\mathcal{A}$:
  Set $f_x = f_x \circ f_{a_x,b_x}$ and $f_t = f_t \circ f_{a_t,b_t}$. Then $\mathcal{S}$ handles msg as in handling Run command.

Note that

$$\begin{aligned} s' &= as + c'v + b = a(cx + t) + c'v + b = a(wc'x/a + t) + c'v + b \\ &= wc'x + at + c'v + b = (wx + v)c' + at + b = x'c' + t'. \end{aligned}$$

Therefore, the above simulation is perfect. ■

Signature Scheme due to Okamoto [O06**].** The digital signature scheme of Okamoto [O06] was employed in the context of designing blind signatures. Here we show that it is BiTR against affine functions. We give a brief description next. Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group as follows:

- $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of prime order q where possibly $\mathbb{G}_1 = \mathbb{G}_2$.
- h_1 (resp. h_2) is a generator of $\mathbb{G}_1$ (resp. $\mathbb{G}_2$).
- ψ is an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ such that $\psi(h_2) = h_1$.
- e is a non-degenerate bilinar map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ where $|\mathbb{G}_T| = p$, i.e.,

$$\forall u \in \mathbb{G}_1\ \forall v \in \mathbb{G}_2\ \forall a, b \in \mathbb{Z} : e(u^a, u^b) = e(u, v)^{ab}.$$

The signature scheme below is secure against a chosen message attack under the Strong Diffie-Hellman assumption [O06]. The signature token is described in Fig. 2.

- Key Generation: Randomly select generators $g_2, u_2, v_2 \in G_2$ and compute $g_1 = \psi(g_2), u_1 = \psi(u_2)$, and $v_1 = \psi(v_2)$. Choose a random $x \in \mathbb{Z}_q^*$ and compute $w_2 = g_2^x$. Verification key is $(g_1, g_2, w_2, u_2, v_2)$. Signing key is x.
- Signature of a message $m \in \mathbb{Z}_q^*$: Choose random $r, s \in \mathbb{Z}_q^*$. The signature is (σ, r, s) where $\sigma = (g_1^m u_1 v_1^s)^{1/(x+r)}$ and $x + r \neq 0 \pmod q$.
- Verification of (m, σ, r, s): Check that $m, r, s, \in \mathbb{Z}_q^*$, $\sigma \in \mathbb{G}_1$, $\sigma \neq 1$, and $e(\sigma, w_2 g_2^r) = e(g_1, g_2^m u_2 v_2^s)$.

Theorem 3. *There is an interface I_{oka} such that ITM M_{oka} in Fig. 2 is $\mathcal{T}_{\mathsf{aff}}$-BiTR parsimonious with I_{oka}.*

M_{sch}: The description of a cyclic group of size q, including a generator g, is embedded in the program as a parameter. The state is $(x, t) \in \mathbb{Z}_q^2$.

Initialization

- Upon receiving a message $(Initialize)$, choose $x, t \in_R \mathbb{Z}_q$ and output g^x.

Message Handling

- Upon receiving a message $(Prove, \perp)$, output $z = g^t$.
- Upon receiving a message $(Prove, c)$, compute $s = cx + t \bmod q$, pick $t \in_R \mathbb{Z}_q$, and output s.

M_{oka}: The description of $\mathbb{G}_1$, $\mathbb{G}_2$, g_2, u_2, v_2, and a collision-resistant hashing function $H : \{0,1\}^n \to \mathbb{Z}_q^*$ are embedded in the program. The state is $x \in \mathbb{Z}_q$.

Initialization

- Upon receiving a message $(Initialize)$, choose $x \in_R \mathbb{Z}_q$, and $g_2, u_2, v_2 \in_R G_2$ and output (g_2, w_2, u_2, v_2).

Message Handling

- Upon receiving a message $(Sign, m)$, Choose random $r, s \in \mathbb{Z}_q^*$ such that $x + r \neq 0 \pmod q$. Compute $\sigma = (g_1^{H(m)} u_1 v_1^s)^{1/(x+r)}$ and output (σ, r, s).

Fig. 2. Schnorr identification M_{sch} and Okamoto signature M_{oka}

Proof. Let $K = M_{oka}$ and $\mathcal{T} = \mathcal{T}_{\mathsf{aff}}$. We show that for any M that calls K as a subprotocol and for all adversaries $\mathcal{A}$ against $(M; K)$, there is $\mathcal{A}', I$ such that for all environments $\mathcal{Z}$ it holds that :

$$\text{IDEAL}_{\mathcal{F}_{twrap}(K,\mathcal{T},\psi),\mathcal{A}^M,\mathcal{Z}^M} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(K),(\mathcal{A}')^{M;I},\mathcal{Z}^M}.$$

Fix M, $\mathcal{A}$, and $\mathcal{Z}$. Let $\mathcal{A}' = \mathcal{A}$. From now on, we will describe the behavior of the interface I. Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted.

For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(K)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(K, \mathcal{T})$. In order to keep the history of tamperings, I maintains a function f_x, which is initialized with the identity function. Wlog, assume the message to be signed is in $\mathbb{Z}_q^*$. The simulator I proceeds as follows:

- Upon receiving $\langle Create, sid, P', P, msg\rangle$ from $\mathcal{A}^M$ on behalf of P':
  I forwards the message back to $\mathcal{A}^M$. When $\mathcal{A}^M$ sends the reply from $\mathcal{F}_{wrap}$, I also forwards the reply back to $\mathcal{A}^M$.
- Upon receiving $\langle Forge, sid, P', P, M', s'\rangle$ from $\mathcal{A}^M$ on behalf of P':
  As with handling $Create$, I forwards the messages to $\mathcal{A}^M$.
- Upon receiving $\langle Run, sid, P, msg\rangle$ from $\mathcal{A}^M$ on behalf of P':
  I forwards the messages back to $\mathcal{A}^M$. When $\mathcal{A}^M$ sends the reply from $\mathcal{F}_{wrap}$, let $\langle sid, P, (\sigma, r, s)\rangle$ be the reply. Let a and b be the coefficients of f_x, i.e., $f_x(z) = az + b$. I computes $\sigma' = \sigma^{1/a}$, $r' = ar - b \bmod q$ and $s' = s$, and sends $\langle sid, P, (\sigma', r', s')\rangle$ to $\mathcal{A}^M$.
- Upon receiving $\langle TamperRun, sid, P, P', f_{a_x,b_x}, msg\rangle$ from $\mathcal{A}^M$:
  Set $f_x = f_x \circ f_{a_x,b_x}$. Then I handles msg as in handling Run command.

Note that

$$\sigma' = \sigma^{1/a} = (g_1^m u_1 v_1^s)^{1/a(x+r)} = (g_1^m u_1 v_1^s)^{1/(ax+ra)} = (g_1^m u_1 v_1^{s'})^{1/(ax+r'+b)} = (g_1^m u_1 v_1^{s'})^{1/(x'+r')}$$

and (r', s') is uniformly distributed in $(\mathbb{Z}_q^*)^2$. Therefore the distribution of (σ', r', s') from I is identical to the distribution from the pass-through interface.

■

5 UC Secure Computation from Tamperable Tokens

In this section we examine the problem of achieving UC-secure computation relying on tamperable (rather than tamper-proof) tokens. Our starting point is the result of Katz [K07], obtaining a UC commitment scheme (and general UC-secure computation) in the $\mathcal{F}_{wrap}(M)$-hybrid for some ITM M (under the DDH assumption). We extend this result to show a UC commitment in the $\mathcal{F}_{wrap}(M')$-hybrid where M' is $\mathcal{T}_{\mathsf{aff}}$-BiTR. Thus, the protocol is secure in the $\mathcal{F}_{twrap}(M', \mathcal{T}_{\mathsf{aff}})$-hybrid model. Along the way we present a generalization of Katz's scheme for building commitment schemes that we call commitments with dual-mode parameter generation. Finally, we examine the OTM token that was introduced by [GKR08] and used by [GIS+10] to achieve unconditional UC-secure computation. We characterize the BiTRness properties of this token.

5.1 Generalizing Katz's Commitment [K07].

Generalizing the result of Katz, we introduce a commitment scheme with *dual-mode parameter generation* (DPG). According to the mode, the parameters, output by DPG, make the commitment scheme unconditionally hiding or extractable. Using this type of commitment, we describe how to implement UC-secure implementation of the ideal commitment functionality.

Definition 5 (Commitment scheme with DPG). *A commitment scheme $\Pi = (Com, Decom)$ that is parameterized by p, has a dual mode parameter generation if there is an ITM M for a two-party protocol $\langle \cdot, M \rangle$ such that*

- *(Normal mode) For any PPT P^*, with overwhelming probability, the output of $\langle P^*, M \rangle$ is a parameter p over which the commitment scheme Π is unconditionally hiding.*
- *(Extraction mode) There is a PPT $\mathcal{S}$ such that $\mathcal{S}^M$ outputs a parameter p and trapdoor t such that the commitment scheme Π with the parameter p is a trapdoor commitment scheme with trapdoor t.*
- *The parameter generated from $\langle P^*, M \rangle$ and that from $\mathcal{S}^M$ are computationally indistinguishable.*

UC-Secure commitment scheme by wrapping DPG. Now we describe our generalized version of the UC-secure commitment protocol given by Katz [K07]. We consider a commitment scheme Π (presented in Fig. 3). The protocol is described in the $\mathcal{F}_{wrap}(M)$ hybrid model (where presumably the ITM M achieves DPG for that commitment scheme).

We briefly sketch the proof of UC-security for the above commitment scheme, assuming that DPG is achieved by the parameter generation. In the real world, due to the property $\mathcal{F}_{wrap}(M)$, it is not possible to rewind M and therefore the parameters are generated in the normal mode. In the ideal world, on the other hand, the simulator emulating the behavior of $\mathcal{F}_{wrap}(M)$ can rewind M and run in the extraction mode. For a corrupted sender, the simulator has to extract the commitment. This can be done by making pS extractable. Then, when the adversary sends a commitment $\langle C_1, C_2, \pi \rangle$, the simulator can extract the message committed to from C_1 using the trapdoor of pS. For a corrupted receiver, the simulator can make the commitment equivocal by causing pR to be extractable. Using the trapdoor that was output along with pR as witness, the simulator can generate a WI proofs π and π' with respect to the condition (2).

We next show the following easy result regarding BiTR properties of the resulting commitment scheme.

Commitment Phase:

1. Each of the sender and the receiver calls $\mathcal{F}_{wrap}(M)$ with a $Create$ message.
2. Each party executes the procedure dual-mode parameter generation with the $\mathcal{F}_{wrap}(M)$. Let pS be the parameter the receiver obtained, and pR be one the sender obtained. The parameters pR and pS are exchanged.
3. The sender commits to a message m by sending $\langle C_1, C_2, \pi \rangle$, where C_1 is a commitment to m based on the parameter pS, C_2 is a statistically-binding commitment to m, and π is WI proof that (1) C_1 and C_2 commits to the same message, or (2) pR was generated in the extraction mode.

Opening Phase:

1. The sender reveals $\langle m, r_2, \pi' \rangle$, where m is the committed message, r_2 is the randomness for C_2, and π' is WI proof that (1) C_1 and C_2 commits to the same message, or (2) pR was generated in the extraction mode.

Fig. 3. Template of the Commitment Scheme Π in the $\mathcal{F}_{wrap}(M)$-hybrid model.

Let $\mathbb{G}$ be the cyclic multiplicative group of size q defined by a safe prime $p = 2q + 1$ and g be a generator of $\mathbb{G}$. The description of $\mathbb{G}$ is embedded in the program. The state is $(r_1, r_2, s_1, s_2) \in \mathbb{Z}_q^4$. It uses a signature ITM K as a subprotocol.

Initialization

- Upon receiving a message $(Initialize)$, call K with $(Initialize)$, sets the state to all 0s and output whatever K outputs.

Message Handling

- Upon receiving a message h_0: Check h_0 is a generator of $\mathbb{G}$. If the checking fails, output $\perp$. Otherwise, pick $r_i, s_i \in_R \mathbb{Z}_q$ and compute Pedersen commitments $\mathsf{com}_i = g^{s_i} h_0^{\mathcal{X}(g_i)}$ for $i = 1, 2$, where $g_i = g^{r_i}$ and the encoding $\mathcal{X}$ is defined as: $\mathcal{X}(\alpha) = \alpha$ if $\alpha > p/2$, $p - \alpha$ otherwise. Output $(\mathsf{com}_1, \mathsf{com}_2)$.
- Upon receiving a message (h, h_1, h_2, x_1, x_2): Check $h, h_1, h_2 \in \mathbb{G}$, $x_1, x_2 \in \mathbb{Z}_q^*$. If the checking fails, output $\perp$. Otherwise, let $g_i = g^{r_i}$ and compute $\hat{g}_i = g_i^{x_i} h_i$ for $i = 1, 2$. Call K with $(Sign, (P, P', p, g, h, \hat{g}_1, \hat{g}_2))$ to get a signature σ. Output $(g_1, g_2, s_1, s_2, \sigma)$. Pick $r_i, s_i \in_R \mathbb{Z}_q$ for $i = 1, 2$.

Fig. 4. DPG-achieving ITM M_{dpg}

Corollary 1. *If an ITM M, achieving DPG for the commitment scheme Π, is $\mathcal{T}$-BiTR, then there exists a UC-secure commitment scheme in the $\mathcal{F}_{twrap}(M, \mathcal{T})$-hybrid.*

5.2 BiTR DPG

Here we present a new dual mode parameter generation for the commitment scheme of Katz [K07], which is BiTR against affine tampering attacks. We briefly recall the basic commitment scheme Π referring to Fig. 3 as instantiated there in [K07]. The main idea is that the parameter p is of the form $(p, g, h, \hat{g}, \hat{h})$. Based on this the commitment C_1 to a bit b is defined as $(g^{r_1} h^{r_2}, \hat{g}^{r_1} \hat{h}^{r_2} g^b)$ for randomly-chosen $r_1, r_2 \in \mathbb{Z}_q$. It is well-known (and easy to check) that if p is a random tuple then this commitment scheme is perfectly hiding; on the other hand if it is a Diffie-Hellman (DH) tuple and $r = \log_g \hat{g} = \log_h \hat{h}$ is known, then b can be efficiently extracted from the commitment. This demonstrates the dual mode properties of the parameter generation.

BiTR DPG-Achieving ITM. We next show there is a DPG-achieving ITM M that is BiTR against affine tampering functions for the commitment scheme. We present this result in Fig. 4. Roughly speaking, the protocol (in the normal mode) generates a random tuple $(g, h, \hat{g}_1, \hat{g}_2)$, by multiplying random numbers g_1 and g_2 (from M_{dpg}) and random numbers h_1 and h_2 (from the token user) — x_1 and x_2 have been introduced to achieve BiTR security. That is, in the normal mode, the probability that the tuple $(g, h, \hat{g}_1, \hat{g}_2)$ is a DH tuple is negligible since $\hat{g}_1$ and $\hat{g}_2$ are uniformly distributed. In the extraction mode, however, $\mathcal{S}$ can rewind

the ITM to cause $(g, h, \hat{g}_1, \hat{g}_2)$ to be a DH tuple. Specifically, $\mathcal{S}$ picks a random DH tuple (g, h', g'_1, g'_2) and, after finding out the values g_1, g_2, rewinds the machine right before the second round and sends $h_i = g'_i / g_1^{x_i}$ for $i = 1, 2$. Under the DDH assumption, parameters from the normal mode and from the extraction mode are indistinguishable.

Theorem 4. *The ITM M_{dpg} in Fig. 4 is $\mathcal{T}^4_{\mathsf{aff}}$-BiTR without any encoding.*

Proof. Let M be the ITM. We show that for any non-uniform PPT $\mathcal{Z}$ and any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}}.$$

Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted.

Fix $\mathcal{Z}$ and $\mathcal{A}$. For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(M, \mathcal{T})$. In order to keep the history of tamperings, $\mathcal{S}$ maintains functions $\{(f_i^r, f_i^s) \mid i = 1, 2\}$, which are initialized with identity functions. The simulator $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.
- Upon receiving $\langle Create, sid, P', P, msg\rangle$ from $\mathcal{A}$ on behalf of P':
  $\mathcal{S}$ forwards the message to $\mathcal{F}_{wrap}$ and sends its response to $\mathcal{A}$ in return.
- Upon receiving $\langle Forge, sid, P', P, M', s'\rangle$ from $\mathcal{A}$ on behalf of P:
  $\mathcal{S}$ forwards the message to $\mathcal{F}_{wrap}$.
- Upon receiving $\langle Run, sid, P, msg\rangle$ from $\mathcal{A}$ on behalf of P':
  - If $msg = h_0$, $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle \mathsf{Run}, sid, P, h_0\rangle$ to get output $\langle sid, P, (\mathsf{com}_1, \mathsf{com}_2)\rangle$. $\mathcal{S}$ sends the output to $\mathcal{A}$ and sets f_i^r and f_i^s to identity functions for $i = 1, 2$.
  - If $msg = (h, h'_1, h'_2, x'_1, x'_2)$, let w_i and v_i be the coefficients of f_i^r (i.e., $f_i^r(z) = w_i z + v_i$) for $i = 1, 2$. $\mathcal{S}$ computes
    $$h_i = h'_i \cdot g^{v_i x'_i}, \;\; x_i = w_i x'_i \bmod q,$$
    for $i = 1, 2$ and sends (h, h_1, h_2, x_1, x_2) to $\mathcal{F}_{wrap}$. Upon receiving $\langle sid, P, (g_1, g_2, s_1, s_2, \sigma)\rangle$ from $\mathcal{F}_{wrap}$, $\mathcal{S}$ computes $g'_i = g_i^{w_i} \cdot g^{v_i}$ and $s'_i = f_i^s(s_i)$ for $i = 1, 2$ and sends $\langle sid, P, (g'_1, g'_2, s'_1, s'_2, \sigma)\rangle$ back to $\mathcal{A}$. Set f_i^r and f_i^s to identity functions for $i = 1, 2$.
- Upon receiving $\langle TamperRun, sid, P, P', (f_{r_1}, f_{r_2}, f_{s_1}, f_{s_2}), msg\rangle$ from $\mathcal{A}$:
  Set $f_i^r = f_i^r \circ f_{r_i}$ and $f_i^s = f_i^s \circ f_{s_i}$ for $i = 1, 2$. Then $\mathcal{S}$ handles msg as in handling Run command.

Let $\hat{g}'_i = (g'_i)^{x'_i} h'_i$ where $g'_i = g^{f_i^r(r_i)}$ for $i = 1, 2$. Note that for $i = 1, 2$

$$\hat{g}'_i = (g'_i)^{x'_i} h'_i = (g_1^{w_i} \cdot g^{v_i})^{x'_i} h_i \cdot g^{-v_i x'_i} = g_1^{x_i} h_i = \hat{g}_i.$$

Therefore the σ is a valid signature on the message $(P, P', g, h, \hat{g}'_1, \hat{g}'_2) = (P, P', g, h, \hat{g}_1, \hat{g}_2)$. The above simulation is perfect. ■

Lemma 1. *ITM M_{oka} in Fig. 2 is a transparent subprotocol of M_{dpg} with respect to I_{oka} in Theorem 3.*

Proof. Let $M = M_{dpg}$, $K = M_{oka}$, and $I = I_{oka}$. We show for any $\mathcal{A}$, there is an $\mathcal{A}'$ such that for all non-uniform $\mathcal{Z}$ it holds that

$$\text{IDEAL}_{\mathcal{F}_{wrap}(K),\mathcal{A}^{M;I},\mathcal{Z}^M} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M;K),\mathcal{A}',\mathcal{Z}}.$$

Fix $\mathcal{A}$ and $\mathcal{Z}$. $\mathcal{A}'$ works as follows:

- $\mathcal{A}'$ works identically with $\mathcal{A}$ except in handling $(M; K)$ tokens.
- When $\mathcal{A}$ sends a message to Create, Forge, Run, or TamperRun[8] $(M; K)$, $\mathcal{A}'$ traps and sends it to $\mathcal{F}_{wrap}(M; K)$. If it was a TamperRun, $\mathcal{A}'$ sends Run $\mathcal{F}_{wrap}(M; K)$ with the same message and remembers the tamper function t. Upon receiving the reply rep from $\mathcal{F}_{wrap}(M; K)$, if K was invoked (i.e., in the second round of DPG), $\mathcal{A}'$ works as follows:
  1. From the reply, $\mathcal{A}'$ extracts the messages exchanged between M and K — in particular, a pair of a message m and a signature σ on it. This is possible, since rep contains the m and σ, according to the DPG protocol.
  2. If $\mathcal{A}$ invoked a TamperRun with t, $\mathcal{A}'$ passes $\langle TamperRun, sid, P, P', t, (Sign, m)\rangle$ to I for preprocessing; Otherwise $\mathcal{A}$ invoked just a Run, $\mathcal{A}'$ passes $\langle Run, P', (Sign, m)\rangle$. From the description of I, we know that it will return the same message. Now, $\mathcal{A}'$ passes $\langle sid, P, \sigma\rangle$ to I for postprocessing. When I sends the modified signature σ', then $\mathcal{A}'$ modifies the signature part of rep.
  3. $\mathcal{A}'$ sends rep to $\mathcal{A}$.

The simulation is perfect. In particular, in the second item above, $\mathcal{A}'$ exactly simulates the messages exchanged among $\mathcal{A}^{M;I}$ and $\mathcal{Z}^M$, and I. ■

Applying the composition theorem (Theorem 1) along with Theorem 3 and 4 and Lemma 1 to the above scheme, we obtain a wholly BiTR token.

Corollary 2. *$(M_{dpg}; M_{oka})$ is $\mathcal{T}^5_{\mathsf{aff}}$-BiTR without any encoding.*

5.3 BiTR One-Time Memory (OTM) Tokens

Following [K07], several works based UC-secure computation on tamper-proof hardware tokens, in various settings. In [GIS+10] Goyal et. al. show (among other things) protocols for general UC-secure computation in the $\mathcal{F}_{wrap}$(OTM)-hybrid model. This hardware token implements a single OT execution, and was introduced by Goldwasser, Kalai, and Rothblum [GKR08] in the context of one-time programs. Specifically, OTM consists of two k-bit strings[9] $\{s_0, s_1\}$. Upon receiving an input bit b, it outputs s_b and updates the state to consist of $\perp$ ("self-destruct").

Theorem 5. *The k-bit OTM protocol is $\mathcal{T}$-BiTR if and only if $\mathcal{T}$ can be written as $\mathcal{T} = (\mathcal{T}_0(s_{j_0}), \mathcal{T}_1(s_{j_1}))$ where each $\mathcal{T}_i : \{0,1\}^k \to \{0,1\}^k$ and $j_0, j_1 \in \{0,1\}$. That is, each of $\mathcal{T}_i$ depends only on one of the secrets.*

Proof. Let M be an OTM as described and $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2)$ with j_0 and j_1. We show that for any non-uniform PPT $\mathcal{Z}$ and any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}}.$$

Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted.

Fix $\mathcal{Z}$ and $\mathcal{A}$. For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(M, \mathcal{T})$. The simulator $\mathcal{S}$ proceeds as follows:

[8] Only the state K can be tampered.
[9] For the [GIS+10] result $k = 1$ (namely two bits) is sufficient.

– $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.
– Upon receiving $\langle Create, sid, P', P, msg\rangle$ from $\mathcal{A}$ on behalf of P':
  $\mathcal{S}$ forwards the message to $\mathcal{F}_{wrap}$ and sends its response to $\mathcal{A}$ in return.
– Upon receiving $\langle Forge, sid, P', P, M', s'\rangle$ from $\mathcal{A}$ on behalf of P':
  $\mathcal{S}$ forwards the message to $\mathcal{F}_{wrap}$.
– Upon receiving $\langle Run, sid, P, msg\rangle$ from $\mathcal{A}$ on behalf of P':
  If $msg = b \in \{0,1\}$, $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle Run, sid, P, msg\rangle$. Let $\langle sid, P, z\rangle$ be the output from $\mathcal{F}_{wrap}$. $\mathcal{S}$ forwards the output $\mathcal{A}$.
– Upon receiving $\langle TamperRun, sid, P, P', (t_0, j_0, t_1, j_1), msg\rangle$ from $\mathcal{A}$:
  If $msg = b \in \{0,1\}$, $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle Run, sid, P, j_b\rangle$. Let $\langle sid, P, z\rangle$ be the output from $\mathcal{F}_{wrap}$. $\mathcal{S}$ sends $\langle sid, P, t_b(z)\rangle$ to $\mathcal{A}$.

It is easy to check $t_b(z) = t_b(s_{j_b})$ is equal to the tampered state s'_b.

■

6 BiTR Protocols against General Classes of Tampering Functions

6.1 BiTR Protocols from Non-Malleable Codes

In this section we will see how the BiTR property can be derived by implementing an integrity check in the form of an encoding ψ. A useful tool for this objective is the notion of non-malleable codes [DPW10] which we recall here: a pair of procedures (E, D) is a non-malleable code with respect to tampering functions $\mathcal{T}$, if there is an algorithm $\mathcal{S}$ such that for all $x \in \{0,1\}^n$ and $t \in \mathcal{T}$, if $x = D(t(E(x)))$, it holds that $\mathcal{S}(t) = \top$ with overwhelming probability, while otherwise $\mathcal{S}(t)$ is statistically (or computationally) close to $D(t(E(x)))$. By encoding the state of a protocol with a non-malleable code it is possible to show the following restatement of Theorem 6.1 of [DPW10] under the BiTR security framework.

Theorem 6 ([DPW10]). *Let $\mathcal{T}$ be a class of tampering functions over $\{0,1\}^m$ and $(E, D, \mathcal{S})$ be a non-malleable code with respect to $\mathcal{T}$, where $E : \{0,1\}^n \to \{0,1\}^m$, $D : \{0,1\}^n \to \{0,1\}^m$ and $\mathcal{S}$ are efficient procedures. Let M be an ITM whose state is of length n. Then M is $(\mathcal{T}, \psi)$-BiTR where $\psi = (E, D)$.*

The above theorem suggests the importance of the problem of constructing non-malleable codes for a given class of tampering functions $\mathcal{T}$. Some positive answers to this difficult question are given in [DPW10] for a class of tampering functions that operate on each one of the bits of the state independently; they also provide a general feasibility result for tampering families of bounded size (with an inefficient construction); an important characteristic of those solutions is relying on the randomness of the encoding. Here we show a different set of positive results by considering the case of **deterministic** non-malleable codes, i.e., the setting where (E, D) are both deterministic functions.

To do that we will utilize a relaxation of non-malleable codes : $(E, D, Predict)$ is called a δ-non-malleable code with distance ϵ if for any $x \in \{0,1\}^n$ and $t \in \mathcal{T}$, it holds that (i) $D(E(x)) = x$, (ii) the probability of the event $D(t(E(x))) \notin \{x, \perp\}$ is at most δ,[10] and (iii) $Predict(\cdot) \in \{\top, \perp\}$, and $|\Pr[D\big(t(E(x))\big) = x] - \Pr[Predict(t) = \top]| \leq \epsilon$. It is easy to see that if ϵ, δ are negligible the resulting code is non-malleable: given that δ is negligible, property (ii) suggests that D will return either the correct value or fail, and thus in case it fails, $Predict(\cdot)$ will return $\perp$ with about the same probability due to (iii). We call δ (resp., ϵ) the crossover threshold (resp., predictability distance).

[10] The tampering t may change the codeword x into another valid codeword.

6.2 Constructing Deterministic Non-Malleable Codes

Inefficient Construction for Any $\mathcal{T}$. Our construction is reminiscent of the procedure that constructs an error-correcting code $\mathcal{C}$ by removing Hamming balls of a certain radius from the codeword space. Let δ, ϵ be the desired crossover threshold and predictability distance respectively. For $u \in \{0,1\}^m$ and $t \in \mathcal{T}$, let $q_{u,t} = \Pr[t(u) = u]$. We say that a set $U \subseteq \{0,1\}^m$ is *one-sided w.r.t.* $\mathcal{T}$ if it holds that $\forall t \in \mathcal{T} : (\forall u \in U : q_{u,t} \leq \epsilon) \vee (\forall u \in U : q_{u,t} \geq 1 - \epsilon)$. The codeword set $\mathcal{C}$ is constructed as follows:

1. Find $u \in \{0,1\}^m$ such that $\{u\}$ is one-sided w.r.t. $\mathcal{T}$. If no such u exists, fail; otherwise, set $\mathcal{C} = \{u\}$.
2. Repeat the following procedure until it stops. Let $L_t = \{u \mid u \in \{0,1\}^m \setminus \mathcal{C} \quad \text{and} \quad (\forall c \in \mathcal{C} : \Pr[t(c) = u] \leq \delta)\}$, and let $L = \cap_{t \in \mathcal{T}} L_t$ (i.e., all strings that cannot be reached by tampering a current codeword, except with probability at most δ). If $L = \emptyset$ then stop; otherwise, pick a $u \in L$ such that $\{u\} \cup \mathcal{C}$ is one-sided and add u to $\mathcal{C}$ (if no such u can be found, stop).

Now, we set $n = \lfloor \log_2 |\mathcal{C}| \rfloor$ and consider $E : \{0,1\}^n \to \{0,1\}^m$ an arbitrary injection from $\{0,1\}^n$ to $\mathcal{C}$. The decoding D is defined as the inverse of E when restricted on $\mathcal{C}$, and $\perp$ everywhere else. We next define $Predict$ as follows. On input t, if $(\forall u \in \mathcal{C} : q_{u,t} \geq 1 - \epsilon)$ holds, then output $\top$; otherwise output $\perp$, since $(\forall u \in \mathcal{C} : q_{u,t} \leq \epsilon)$ will hold (no other option is possible, since $\mathcal{C}$ is one-sided by construction).

The rate of the constructed code is n/m, the time-complexity of constructing E, D is $2^{\mathcal{O}(m+n)}|\mathcal{T}|$ and the time complexity of $Predict(\cdot)$ is $\mathcal{O}(2^m)$.[11] Regarding this construction we have:

Lemma 2. *Fix any class of functions $\mathcal{T}$. The above code $(E, D, Predict)$ is δ-non-malleable w.r.t. $\mathcal{T}$ and distance ϵ. Moreover, if there exists such code with rate > 0, then such a code is produced by the above procedure.*

Proof. Let $t \in \mathcal{T}$. For any x it holds that the event $t(E(x)) \notin \{x, \perp\}$ (cross-over) happens with probability at most δ by construction of $\mathcal{C}$.

On the other hand, let $x \in \{0,1\}^n$. Denote $u = E(x)$ and we have that $D(t(E(x))) = x$ suggests that $t(u) = u$. This event happens with probability $q_{u,t}$ (but note that $Predict$ is only given t, and is independent of u - this is where the one-sidedness of $\mathcal{C}$ is used). We consider the two possible cases. If $\forall u \in \mathcal{C} : q_{u,t} \leq \epsilon$ (i.e., t is likely to corrupt the encoding), $Predict$ is defined to output 0, while $\Pr[D(t(E(x))) = x] = q_{u,t} \leq \epsilon$. If $\forall u \in \mathcal{C} : q_{u,t} \geq 1 - \epsilon$ (i.e., t is close to the identity function), $Predict$ is defined to output 1, while $\Pr[D(t(E(x))) = x] = q_{u,t} \geq 1 - \epsilon$. In both cases we have the needed

$$|\Pr[D\big(t(E(x))\big) = x] - \Pr[Predict(t) = 1]| \leq \epsilon.$$

■

Discussion: When does a deterministic non-malleable code exist? The basic idea of the construction above was to search for a one-sided set of codewords and use it to define the non-malleable code. The

[11] In the above description, we assumed the probabilities $q_{u,t}$ and $\Pr[t(c) = u]$ are known. If they are not known, they can be estimated using standard techniques. In particular, to evaluate the probability of an event A, repeat k independent experiments of A and denote the success ratio of the k experiments as $\hat{p}$. Let X_i be the probability that the i-th execution of the event A is successful. The expected value of $Y = \sum_{i=1}^{k} X_i$ is $k \cdot p$. Using the Chernoff bound it follows that $|\hat{p} - p| \leq 1/N$ with probability $1 - \gamma$ provided that $k = \Omega(N^2 \ln(\gamma^{-1}))$.

necessity of one-sidedness is easy to see since if the property fails, i.e., $\epsilon < q_{u,t} < 1 - \epsilon$ for some t and u, the requirement on $Predict$ cannot hold in general.[12] We now provide two illustrative examples and discuss the existence (and rate) of a deterministic non-malleable encoding for them.

Example 1: Set Function. If $\mathcal{T}$ contains a function t that sets the i-th bit of $u \in \{0,1\}^m$ to 0, it follows that the code $\mathcal{C}$ we construct must obey that either all codewords have the i-th bit set to 0 or all of them have the bit set to 1. This means that any bit setting function cuts the size of the code $|\mathcal{C}|$ by half. There is no non-malleable code when the collection $\mathcal{T}$ contains Set functions for every bit position (consistent with the tampering attack of [GLM+04] using Set functions).

Example 2: Differential Fault Analysis [BDL01]. Consider a single function t which flips each 1-bit to a 0-bit with probability β. Consider a code $\mathcal{C} \subseteq \{0,1\}^m$ for which it holds that all codewords in $\mathcal{C}$ have Hamming distance at least r between each other and $0^m \in \mathcal{C}$. Then it is easy to see that δ, the probability of crossover, is at most β^r. Further, now suppose that t is applied to an arbitrary codeword u in $\mathcal{C}$ other than 0^m. We observe that the number of 1's in u is at least r (otherwise it would have been too close to 0^m). It follows that t will change some of these 1's to 0's, with probability at least $1 - (1-\beta)^r$. It follows that we can predict the effect of the application of t with this probability when we restrict to codewords in $\mathcal{C} \setminus \{0^m\}$.

Claim. Any code $\mathcal{C}$ over $\{0,1\}^m$ with minimum distance r that contains 0^m allows for a β^r-non-malleable code with $(1-\beta)^r$ for t using the code $\mathcal{C} \setminus \{0^m\}$.

Using an asymptotically good code we can essentially retain the same message-rate.

We can extend the above to the case when a compositions of t are allowed. Note that a sequence of a applications of t, flips each 1-bit to a 0-bit with probability $\beta + (1-\beta)\beta + \ldots + (1-\beta)^{a-1}\beta = 1 - (1-\beta)^a$. The encoding now has crossover $(1 - (1-\beta)^a)^r \leq e^{-(1-\beta)^a r}$. Thus, from $e^{-(1-\beta)^a r} \leq \delta$, we obtain $r \geq (1/(1-\beta))^a \ln(1/\delta)$, i.e., when β is bounded away from 1, the minimum distance of the code grows exponentially with a.

Efficient Construction for Localized $\mathcal{T}$. Now, we show a simple way to use the (inefficient) construction above with constant rate and any cross-over $\delta < 1/2$, to achieve an efficient construction with negligible cross-over (and thus, a BiTR protocol), when the class contains only functions that can be split into independent tampering of local (i.e., logarithmically small) blocks. Here we consider a tampering class $\mathcal{T}$ of polynomial size.

Theorem 7. *Let k be a security parameter. Let $\mathcal{T}_i$ be a class of tampering functions over $\{0,1\}^{m'}$ for $i = 1, \ldots, \ell$ where $m' = \mathcal{O}(\log k)$. Let $\mathcal{T}$ be a class of functions over $\{0,1\}^m$ defined as $\mathcal{T} = \mathcal{T}_1 \times \cdots \times \mathcal{T}_\ell$ (i.e., $m = \ell m'$). Suppose that there exist deterministic δ-non-malleable codes w.r.t. $\mathcal{T}_i$ with rate r and $\delta < 1/2$ for $i = 1, \ldots, \ell$. Then there exists a deterministic non-malleable code w.r.t. $\mathcal{T}$ with rate $\geq r/2$ as long as $\ell \leq 2^{m'r}$ and $\ell = \omega(\log k)$.*

Proof. We show a construction of a non-malleable code w.r.t. $\mathcal{T}$. Let $n' = rm'$ and $n = mr/2$. Note that $\ell = m/m' = 2n/n'$. Let $q = n/n' = \ell/2$.

Encoding. Given an input $x \in \{0,1\}^n$, the encoding proceeds as follows:

1. Parse the given input x as $x_1, \ldots, x_q \in \{0,1\}^{n'}$.

[12] For probabilistic non-malleable codes this can be dealt with, as long as $q_{E(x),t}$ is independent of x — i.e., for any possible encoded value, there is the same probability that t corrupts it.

2. Apply a Reed-Solomon code to $x_1, \ldots, x_q$, and obtain a codeword $y_1, \ldots, y_\ell$ such that

$$y_j = \sum_{i=0}^{q-1} (\alpha_j)^i x_{i+1}$$

where $\alpha_1, \ldots, \alpha_\ell$ are distinct elements. Note it has to be the case that $2^{n'} > \ell$.

3. For $j = 1, \ldots, \ell$, encode each $y_j \in \{0,1\}^{n'}$ to $z_j = E_j(y_j)$, where E_j is the encoding function of the δ-non-malleable code w.r.t. $\mathcal{T}_i$ with $negl(k)$ predictability distance.

Decoding. To decode a codeword $(z_1, \ldots, z_\ell)$, for $i = 1, \ldots, \ell$, compute $y_i = D_j(z_i)$ for $i = 1, \ldots, \ell$, where D_j is the decoding function of the δ-non-malleable code w.r.t. $\mathcal{T}_i$ with $negl(k)$ predictability distance. If one of those individual decodings fails then decoding fails. Otherwise, check to see if the points $\{(\alpha_i, y_i)\}_{i=1}^{\ell}$ lie on a polynomial of degree less than q. If this is the case output the polynomial's coefficients $(x_1, \ldots, x_q)$; otherwise the decoding fails.

Now, consider the family of tampering functions $\mathcal{T}$. We show the crossover parameter of the above construction is $negl(k)$. Given that the action in each coordinate is independent the probability of crossover is the probability of switching $\ell - q + 1$ coordinates. This probability via the Chernoff bound can be made exponentially small assuming that $\delta < 1/2$. Predictability is easily satisfied by checking if the output of Predict for each block is 1. ■

Putting this with Theorem 6, we get the following.

Corollary 3. *Under the conditions and parameter choices of Theorem 7, any protocol with state of length at most $mr/2$ can be made $\mathcal{T}$-BiTR as long as $\ell \leq 2^{m'r}$ and $\ell = \omega(\log k)$.*

References

[ADW09] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[AGV09] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[BB05] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[BDL01] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

[BG10] Z. Brakerski and S. Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In *CRYPTO*, pages 1–20, 2010.

[BS97] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[C01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CF01] R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CGGM00] R. Canetti, O. Goldreich, S. Goldwasser, and S. Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000.

[CGS08] N. Chandran, V. Goyal, and A. Sahai. New constructions for uc secure computation using tamper-proof hardware. In *EUROCRYPT*, pages 545–562, 2008.

[DGK+10] Y. Dodis, S. Goldwasser, Y. T. Kalai, C. Peikert, and V. Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[DKL09] Y. Dodis, Y. T. Kalai, and S. Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DNW08] I. Damgård, J. B. Nielsen, and D. Wichs. Isolated proofs of knowledge and isolated zero knowledge. In *EUROCRYPT*, pages 509–526, 2008.

[DP10] Y. Dodis and K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, pages 21–40, 2010.

[DPW10] S. Dziembowski, K. Pietrzak, and D. Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[FKPR10] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[FRR+10] S. Faust, T. Rabin, L. Reyzin, E. Tromer, and V. Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[GIMS10] V. Goyal, Y. Ishai, M. Mahmoody, and A. Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *CRYPTO*, pages 173–190, 2010.

[GIS+10] V. Goyal, Y. Ishai, A. Sahai, R. Venkatesan, and A. Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GKR08] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GLM+04] R. Gennaro, A. Lysyanskaya, T. Malkin, S. Micali, and T. Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

[GR10] S. Goldwasser and G. N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[IPSW06] Y. Ishai, M. Prabhakaran, A. Sahai, and D. Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[JV10] A. Juma and Y. Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[K07] J. Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.

[K10] V. Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In *TCC*, pages 327–342, 2010.

[KJJ99] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[KV09] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[MR04] S. Micali and L. Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[MS08] T. Moran and G. Segev. David and goliath commitments: Uc computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, pages 527–544, 2008.

[NS09] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[O06] T. Okamoto. Efficient blind and partially blind signatures without random oracles. In *TCC*, pages 80–99, 2006.

[P09] K. Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[QS01] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-SMART*, pages 200–210, 2001.

[S91] C.-P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[SMY09] F.-X. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, pages 443–461, 2009.