Practical Non-Malleable Codes from L-more Extractable Hash Functions

Citation for published version:

Digital Object Identifier (DOI):
10.1145/2976749.2978352

Link:
Link to publication record in Edinburgh Research Explorer

Document Version:
Publisher's PDF, also known as Version of record

Published In:
Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security

General rights
Copyright for the publications made accessible via the Edinburgh Research Explorer is retained by the author(s) and / or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights.

Take down policy
The University of Edinburgh has made every reasonable effort to ensure that Edinburgh Research Explorer content complies with UK legislation. If you believe that the public display of this file breaches copyright please contact openaccess@ed.ac.uk providing details, and we will remove access to the work immediately and investigate your claim.
Practical Non-Malleable Codes from $\ell$-more Extractable Hash Functions

Aggelos Kiayias  
University of Edinburgh  
akiayias@inf.ed.ac.uk

Feng-Hao Liu  
Florida Atlantic University  
fenghao.liu@fau.edu

Yiannis Tselekounis*  
University of Edinburgh  
tselekounis@sians.org

ABSTRACT

In this work, we significantly improve the efficiency of non-malleable codes in the split state model, by constructing a code with codeword length (roughly) $|s| + 9k$, where $|s|$ is the length of the message, and $k$ is the security parameter. This is a substantial improvement over previous constructions, both asymptotically and concretely.

Our construction relies on a new primitive which we define and study, called $\ell$-more extractable hash functions. This notion, which may be of independent interest, is strictly stronger than the previous notion of extractable hash by Goldwasser et al. (Eprint ’11) and Bitansky et al. (ITCS ’12, Eprint ’14), yet we can instantiate it under the same assumption used for the previous extractable hash function (a variant of the Knowledge of Exponent Assumption).

Keywords

Non-malleable codes, hash functions, split-state model

1. INTRODUCTION

Non-malleable codes were introduced by Dziembowski et al. [29] as a relaxation of error correction and error detection codes. They provide security in the following sense: any modified codeword decodes to the original message or to a completely unrelated one, with overwhelming probability. Non-malleability is defined through a simulation-based definition, which informally states that, for any tampering function $f$, we require the existence of a simulator that simulates the tampering effect, by only inspecting $f$, i.e., without making any assumptions on the distribution of the encoded message.

Various applications of non-malleable codes have been proposed, such as CCA secure encryption schemes [21], non-malleable commitments [5], and most notably, their application against malicious modification attacks, also known as tampering attacks. Indeed, using non-malleable codes to secure implementations against tampering attacks was the motivation in the original work by Dziembowski et al. [29]. Due to their important application, constructing non-malleable codes has drawn a lot of attention, as we elaborate below.

The split-state model [29, 40]. Ideally, we would like to achieve non-malleability against arbitrary function classes, yet, this task is not achievable, as it is also not achievable in the case of error correction/detection codes. As discussed in [29], assuming a tampering function $f$ that computes the decoding of the codeword $c = \text{Enc}(s)$, where $s$ is the private message, and computes $\tilde{c} = \text{Enc}(s + 1)$, we receive a tampered codeword, $\tilde{c}$, that decodes to a message, highly related to the original one. Therefore, no secure construction can exist against any function class that contains $f$, which concludes that, restricting the function class, is inherent.

Motivated by the above, various function classes have been studied, and in particular, the split-state function class has been identified and extensively studied in the literature. Briefly speaking, in the split-state model, private memory is split in two parts, $L$, $R$, and the attacker may apply any function $f = (f_1, f_2)$ that results in a tampered memory equal to $(f_1(L), f_2(R))$. This is a plausible model since in many cases sensitive data may be split in two storage devices that are physically separated. Note that the model can generalize to multiple split states, with the two-state variant being the hardest to achieve; we only consider the two state variant in this paper.

Broadly speaking, (explicit) constructions of non-malleable codes in the split-state model can be categorized into information-theoretic and computational. In a recent breakthrough result [4], Aggarwal et al. provide the first polynomial-time, information-theoretic, non-malleable code for multi-bit messages, thus significantly improving over the work of [28], which only supports single-bit messages. The encoder produces codewords of length $O((|s| + k)^7)$, where $|s|$

---

*Work performed while at the National and Kapodistrian University of Athens.
Research supported by ERC project CODAMODA, # 259152 and H2020 Project Panoramix # 653497.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored.
For all other uses, contact the owner/author(s).

CCS’16 October 24–28, 2016, Vienna, Austria
© 2016 Copyright held by the owner/author(s).
DOI: http://dx.doi.org/10.1145/2976749.2978352
denotes the length of the encoded message, $s$, and $k$ is the security parameter.\footnote{The result of [4] can be further improved assuming specific conjectures.} Later Aggarwal et al. [3] proposed another construction that achieves codeword length roughly $O(|s|)$ (for sufficiently large $|s|$).ootnote{The hidden constants might be “astronomical” as they depend on results in additive combinatorics, as pointed out in the conclusion of their work [3].} In the computational setting, Liu and Lysyanskaya [40] construct a non-malleable code using cryptographic tools such as leakage resilient public-key encryption [43], and robust non-interactive zero-knowledge (NIZK) proofs [26]. The rate of their construction is not given in the original paper and a textbook instantiation with public-key encryption combined with NIZKs, would not yield a rate 1 code; however, using state of the art tools, we can provide a better instantiation of [40], with codeword length $|s| + O(k^2)$, see Table 1.1. Recently, Aggarwal et al. [2] presented a compiler that transforms any low rate, non-malleable code, to a rate 1, computationally secure, non-malleable code. The underlying encoding must satisfy a notion, strictly stronger than non-malleability, called augmented non-malleability, which, as it is stated in [2], can be satisfied by the construction of [4]. Thus, by instantiating the compiler of [2] with the construction of [4], the codeword’s length becomes $|s| + O(k^2)$.

Although the above constructions achieve “rate 1” asymptotically, i.e., the ratio of message to codeword length is 1, as the message length, $|s|$, goes to infinity, in practice, the induced overhead can still be too large, when considering short messages (e.g., a 160-bit cryptographic key), even without counting the potentially large hidden constants in the asymptotic notation. Thus, even though the problem of “optimal-rate” has been solved in theory, it is still unclear what the practical implications of those constructions are. Given the current state of the art, as discussed above, constructing codes with very small overhead, including the hidden constant, remains still one of the most important open questions in the area. Note, that the natural lower bound for code length is merely $|s| + k$, and none of the known, computational or information-theoretic, constructions, match it, even asymptotically.

### 1.1 Our Results

In this work, we tackle the challenge to construct truly efficient non-malleable codes in the split-state model. To achieve this goal, we introduce a new cryptographic primitive, called $\ell$-more extractable hash function family, and then we construct an efficient code, using our new tool. Our approach is modular: first we propose and formalize $\ell$-more extractable hash function families, and then we demonstrate their application to non-malleable codes.

Briefly speaking, $\ell$-more extractable hash function families capture the idea that, if an adversary, given $\ell$ hash values $v_1, \ldots, v_\ell$, produces a new valid hash value $\tilde{v}$, then it must know a pre-image of $\tilde{v}$. This is a generalization of the notion of extractable hash functions by Bitansky et al. [10] and Goldwasser et al. [49], which corresponds to the $\ell = 0$ case (i.e., the adversary gets no access to valid hash values, prior to producing its own value), and is somewhat reminiscent of the strengthening of simulation-soundness in the context of zero-knowledge proofs [48]. Our generalization is strict: we prove the following (informally stated):

**Theorem 1.1 (Informal).** Extractable hash $\not\Rightarrow$ 1-more extractable hash.

The subtlety comes from the fact that the $\ell$-more attacker might get an “unfair advantage” in producing a valid hash value, for which it does not possess a pre-image, because of the $\ell$ additional inputs; e.g., by modifying the $v_i$’s in some suitable way. Indeed, we show that the extractable hash function family of Bitansky et al. [10] is easily malleable, and thus exploitable by “1-more” attackers. This demonstrates that our new notion of $\ell$-more extractability is strictly stronger than the previous one.

Our next step is to achieve such a stronger notion. We show, somewhat surprisingly, that the stronger notion can be achieved under the same assumptions used by the construction of Bitansky et al. [10], i.e., a variant of the Knowledge of Exponent Assumption (KEA) and DLOG. Thus, we conclude that, even though $\ell$-more extractability is strictly stronger, KEA and DLOG are still sufficient to achieve it.

**Theorem 1.2 (Informal).** DLOG and (a variant of) KEA imply $\ell$-more extractable hash.

We remark that KEA is non-falsifiable (cf. [42]), and it is indeed a strong assumption. However, one can argue that non-falsifiability might be inherent for extractable hash functions, and thus $\ell$-more extractability. We recall that Bitansky et al. [10] showed that, extractable hash function families imply succinct non-interactive argument of knowledge (SNARK), and Gentry and Wichs [35] showed that SNARK is unlikely to be constructed based on falsifiable assumptions. Thus, non-falsifiable assumptions are likely to be inherent for achieving ($\ell$-more) extractability. We note that some variants of KEA were shown to contradict (public-coin) differing-inputs obfuscation and indistinguishability obfuscation [12, 13]; the variant we use is suitably defined to circumvent this contradiction.

Next, we construct non-malleable codes using $\ell$-more extractable hash functions. The crux of our methodology is to adapt the “public-key-encrypt-and-prove” method of [40], using our new $\ell$-more extractable hash, yielding effectively a “(one-time-symmetric-key-encrypt-and-hash)” approach for obtaining non-malleable codes. In particular, we prove the following (informally stated):

**Theorem 1.3 (Informal).** $\ell$-more extractable hash (with some additional properties) implies non-malleable codes in the split-state model.

Our scheme produces codewords of length $|s| + 9 \cdot k + 2 \cdot \log^2 (k)$. In Table 1.1 we compare our construction with the current state of the art on the split-state setting. Our scheme is truly efficient in terms of codeword length, and it is one order of magnitude better than the combination of [40] + [2] + [43] + [37], which is the most competitive scheme that can be constructed,\footnote{For the sake of this comparison, we instantiate [40] with the efficient zero-knowledge proofs of [37] and the leakage resilient public-key encryption of [43]; moreover we observe that the resulting code is compatible with the compiler of [2] (it satisfies “augmented non-malleability”, a property defined in the latter paper) and thus we can make the resulting system rate 1. This provides codeword length $|s| + O(k^2)$, cited in Table 1.1.} based on the current state of the
1.2 Technical Overview

Concepts of extractability and $\ell$-more extractability. Informally, a family of functions, $\mathcal{F}$, is extractable, if for a uniform $h \in \mathcal{F}$, there exists an extractor $E$, such that for any algorithm $A$, there exists an extractor $E_A$, such that if $A$, produces some $v \in \text{image}(h)$, $E_A$, outputs $s$, such that $h(s) = v$. Clearly, such families are interesting only if they possess some sort of hardness property, like one-wayness, otherwise the problem is trivial.

In this work, we introduce the notion of $\ell$-more extractable hash function families, for which the extractability property holds, even if $A$, is given access to $\ell$ valid hash values. Even though $\ell$-more extractability looks similar to extractability (0-more extractability in our definition), we provide a separation between those two primitives. Before explaining further details, we first recall the underlying assumption, $\ell$-KEA, and the construction of Bitansky et al. [10].

$\ell$-KEA and the extractable hash function family of [10]. Assuming a group $G$, of prime order $p$, the Knowledge of Exponent Assumption (KEA), introduced by Damgård [25], states the following: any adversary that is given a generator, $g$, of $G$, and a random group element $g^a$, produces the pair $(g^{a_1}, g^{a_2})$, only if it “knows” the exponent $a$. The assumption was later extended by [8, 38], by requiring that, given $g^{a_1}$, $g^{a_1 r}$, $g^{a_2}$, it is infeasible to produce $v = g^{a_1 + a_2 + r}$ and $v^a$, without “knowing” $s_1$, $s_2$. This assumption, generalized for $t = \text{poly}(\log |G|)$ pairs $g^{a_1}$, $g^{a_1 r}$, is referred to as $t$-KEA by [10].

An element from the hash function family of [10] is described by the pair $(g^{a}, g^{ar})$, for uniformly random vector $r$, and element $a$. Note that, $g^{a}$ denotes the value $(g^{a_1}, \ldots, g^{a_t})$, where $r = (r_1, \ldots, r_t)$. The hash of a message $s = (s_1, s_2)$, is the pair $(g^{(a,s)}, g^{(r,s)})$, where $(a,s)$ denotes the inner product of $r$, $s$. It is not hard to see that the hash value can be computed efficiently given the message and the description of the hash function, and assuming the $t$-KEA, the above hash function family is extractable, or in our terminology, 0-more extractable. As we argue in the next paragraph, this family is not 1-more extractable, and thus, extractability does not imply $\ell$-more extractability.

1-more Extractable Collision Resistant Hash (ECRH).

Suppose the adversary receives a hash value $v = h(s) = (g^{(a,s)}, g^{(r,s)})$, for some unknown message $s$, and then computes $v' = v^x = (g^{(a,tx)}, g^{(r,tx)})$, for some non-zero $x$, of its choice. Clearly, the new hash value $v'$ equals $h(xs)$, and thus, it is valid. Then, assuming an extractor for the current family, under the “1-more” setting, we can retrieve the original message $s$, by first extracting $zs$ and then dividing it by $x$. This idea can be turned into a DLOG solver, and thus, assuming the hardness of DLOG with respect to $G$,

we show in Lemma 3.5, that the above construction is not 1-more extractable.

Next we present our strategy for constructing 1-more ECRH. Our main observation is that, even though the above hash function family is malleable, the modified hash value, $v'$, has some structure: it is the hash value of the message yielded after applying an affine transformation on the original message, $s$, (in the above case, the affine transformation was $x \cdot s$). Interestingly, we show that under the $t$-KEA, applying an affine transformation is the only thing the adversary can do! In particular, we show that, if the adversary outputs a valid, new hash value, $v'$, then there exists an extractor that extracts an affine transformation on the underlying message. So, in order to make the hash non-malleable (and then 1-more extractable), we first encode $c \leftarrow \text{Enc}(s)$ using a non-malleable code against affine functions, and then we compute $v = (g^{(a,c)}, g^{(a,c')})$. This approach can be viewed as a computational analogue of a non-malleable reduction, as previously used by [4], and then formally presented by [3] (both are in the information-theoretic setting).

It turns out that, in order to apply the methodology described above, a slightly stronger flavor of non-malleability is required for the underlying code, which we formalize as randomizable simulatable non-malleable codes. Below, we briefly discuss this notion and we give the main idea of the proposed scheme.

Randomness simulatable NM codes against affine tampering. This notion of non-malleability is stronger than the standard one, in the sense that, besides simulating the pre-image of the tampered codeword, $\tilde{s}$, the simulator, also produces the randomness of the encoder, $\tilde{t}$, such that the encoding of $\tilde{s}$ with randomness $\tilde{t}$, produces the tampered codeword. The main idea of our construction method is given in the next paragraph.

For any message $s$, the encoder secret shares $s$ into $(s_1, s_2)$, using a two-out-of-two, additive, secret sharing scheme, and outputs $e = (s_1, s_2, s_1^2, s_2^2)$. Then, for any codeword $c = (s_1, s_2, s_1', s_2')$, decoding proceeds as follows: if $s_i^2 = s_i'$, for $i \in \{1, 2\}$, the decoder outputs $s_1 + s_2$, otherwise, it outputs $\perp$. An affine tampering function, $f$, against the code is described by the pair $(b, d)$, and the application of $f$ on a codeword $c$, yields the codeword $d \cdot c + b$. We prove security of the above code by considering the following cases (roughly). If $d = 0$, then the tampered codeword is completely overwritten by $b$, and clearly, the output of the decoder depends only on $b$. If $d \neq 0$, then, we argue that, either the attack leaves the codeword intact, i.e., $d = 1$, $b = 0$, or the decoding of the tampered codeword is $\perp$, with overwhelming probability.

In Section 5.2, we formally define randomness simulatable, non-malleable codes, and prove security for the proposed scheme. It is worth to point out that the idea of constructing a NM-code for affine functions, as an intermediate step for providing split-state codes, was also followed by [4], still, our technique differs significantly, and their code does not directly satisfy our requirements. Moreover, in [22] the authors construct AMD codes, still their notions are slightly different and do fit in our framework.

NM codes against split-state tampering. Our construction of non-malleable codes is inspired by the one of Liu and Lysyanskaya [40], so we first recall their construction. To encode a message $s$, their encoder outputs $(sk, (pk,$
their scheme can only encode single-bit messages. Subsequent constructions for multi-bit messages are discussed in the previous section. Non-malleable codes for other function classes have been extensively studied, e.g., bit-wise independent tampering [29], bounded-size function classes [32], the k-split setting [18], block-wise tampering [16, 20], and bounded depth and fan-in circuits [6]. The work of [3] develops beautiful connections among different function classes.

KEAs and previous work. In [25], Damgård introduces KEA to construct a CCA-secure encryption scheme. In [8, 38], the authors extend the assumption of [25], and construct three-round, zero-knowledge arguments. Abe and Fehr [1] construct the first perfect NIZK for NP with adaptive soundness, by extending the assumption of [8]. Prabhakaran and Xue [47] constructed statistically-hiding sets for trapdoor DDH groups [27], by introducing a new knowledge assumption. Gennaro et al. [34] proved that a modified version of the Okamoto-Tanaka key-agreement protocol [44] satisfies perfect forward secrecy against fully active attackers, by introducing a new knowledge assumption. In [9–11, 33, 36], the authors construct succinct, non-interactive, arguments of knowledge (SNARKs), and NIZKs, while in [41], Mie presents a private information retrieval (PIR), scheme. In [14, 15, 24], Canetti and Dachrouf provide an extensive study on extractable functions. In [45], Parno et al. show how to perform verifiable computation, efficiently.

In [12, 13], the authors show that, assuming indistinguishability obfuscation [7], extractable one-way functions, and thus ECRHs, does not exist against adversaries receiving arbitrary, polynomial-size, auxiliary input, if the extractor is fixed before the attacker’s auxiliary input. On the other hand, they show that, under standard assumptions, extractable one-way functions, may exist against adversaries with bounded auxiliary input.

In this work, and as it is suggested by [12], we consider individual auxiliary, i.e., we allow the auxiliary info of the extractor to depend on the attacker’s auxiliary info, and therefore, we do not contradict the impossibility results of [12, 13].
2. PRELIMINARIES

In this section we present basic primitives and notation that we use in our constructions.

**Definition 2.1 (Notation).** $\mathbb{N}^+, \mathbb{R}^+$, denote the set of positive natural and real numbers, respectively. For $t \in \mathbb{N}^+$, $[t]$ is the set $\{1, \ldots, t\}$. For vectors $\mathbf{x}, \mathbf{y}, (\mathbf{x}, \mathbf{y})$ is the inner product of $\mathbf{x}$, $\mathbf{y}$, and $|\mathbf{x}|$ is the $i$-th coordinate of $\mathbf{x}$. For strings $x, y, x[y]$ is the concatenation of $x, y$, and $|x|$ denotes the length of $x$. For a distribution $D$ over a set $\mathcal{X}$, $x \sim D$, denotes sampling an element $x \in \mathcal{X}$, according to $D$. $x \in \mathcal{X}$, denotes sampling a uniform element $x$, from $\mathcal{X}$, and $U_{\mathcal{X}}$ denotes the uniform distribution over $\mathcal{X}$. The statistical distance between two random variables $X, Y$, with range $\mathcal{D}$, is denoted by $\Delta(X, Y)$, $\Delta(X, Y) = \frac{1}{2} \sum_{u \in \mathcal{D}}|Pr[X = u] - Pr[Y = u]|$. Moreover, $\approx$ and $\cong$ denote statistical and computational indistinguishability, respectively. A function $f : \mathbb{N} \rightarrow \mathbb{R}^+$ is negligible, if for every positive polynomial poly($\cdot$), and all sufficiently large $k$, $f(k) \leq 1/poly(k)$, and negl$(k)$ denotes an unspecified, negligible function, in $k$. For a random variable $X$, $H_{\infty}(X)$ and $H_{\infty}(X)$ denote the min-entropy, and average min-entropy, of $X$, respectively. Finally, for any element $g$ and vector $s = (r_1, \ldots, r_t)$, $g^s = (g^{r_1}, \ldots, g^{r_t})$.

Below, we define coding schemes, based on the definitions of [29, 40].

**Definition 2.2.** (Coding scheme in the common reference string (CRS) model [40]) $\kappa(\cdot, \cdot)$-coding scheme in the CRS model, $\kappa, \nu \in \mathbb{N}$, is a triple of algorithms $(\text{Init}, \text{Enc}, \text{Dec})$ such that: $\text{Init}$ is a randomized algorithm which receives $1^k$, where $k$ denotes the security parameter, and produces a common reference string $\Sigma \in \{0, 1\}^{poly(k)}$, and $(\text{Enc}(1^k, \Sigma, \cdot), \text{Dec}(1^k, \Sigma, \cdot))$ is a $(\kappa, \nu)$-coding scheme, $\kappa, \nu = poly(k)$.

For brevity, $1^k$ will be omitted from the inputs of Enc and Dec. In the full version of the paper we provide the standard definitions of coding schemes and non-malleability. Now we state the definition of strong non-malleability in the CRS model based on the definitions of [29, 40].

**Definition 2.3.** (Strong non-malleability in the CRS model, [29, 40]) Let $(\text{Init}, \text{Enc}, \text{Dec})$ be a $(\kappa, \nu)$-coding scheme in the common reference string model, and $F$ be a family of functions $f : \{0, 1\}^\nu \rightarrow \{0, 1\}^\nu$. For any CRS $\Sigma$, $f \in \mathcal{F}$ and $s \in \{0, 1\}^s$, define the tampering experiment

$$\text{Tamper}_s^{\Sigma, f} \equiv \left\{ \begin{array}{l} c \leftarrow \text{Enc}(\Sigma, s), \tilde{c} \leftarrow f^c(e), \tilde{s} = \text{Dec}(\Sigma, \tilde{c}) \quad \text{Output same} \quad \text{if } \tilde{c} = c, \text{ and } \tilde{s} \text{ otherwise.} \end{array} \right\}$$

which is a random variable over the randomness of Enc and Dec. The coding scheme $(\text{Init}, \text{Enc}, \text{Dec})$ is strongly non-malleable with respect to the function family $\mathcal{F}$, if for each $f \in \mathcal{F}$ and any $s_0, s_1 \in \{0, 1\}^s$

$$\left\{ \left( \Sigma, \text{Tamper}_s^{\Sigma, f} \right) \right\}_{s \in \mathbb{N}} \approx \left\{ \left( \Sigma, \text{Tamper}_{s_1}^{\Sigma, f} \right) \right\}_{s \in \mathbb{N}},$$

where $\Sigma \leftarrow \text{Init}(1^k)$, and “$\approx$” may refer to statistical, or computational, indistinguishability, with parameter $k$.

According to the standard definition of non-malleability, the decoding procedure is not randomized, however, as it is suggested by Ball et al. [6] Dec may be randomized.

Next we state the $t$-variant, due to [10], of the Knowledge of Exponent assumption (KEA), [8, 25, 38], with individual auxiliary inputs for adversary and extractor, which is known not to contradict the impossibility results of [12, 13].

**Assumption 2.4 (t-KEA assumption).** Let $t \in \mathbb{N}$. There exists a group generation algorithm $G$, such that for any pair $(g, g')$ sampled according to $G(1^k)$, where $\mathbb{G}$ is a group of prime order $p \in (2^{k-1}, 2^k)$, the following holds: for any PPT algorithm $A$ with auxiliary input aux$_A \in \{0, 1\}^{poly(k)}$, there exist PPT extractor $E_A$ with auxiliary input aux$_E \in \{0, 1\}^{poly(k)}$, such that for all sufficiently large $k \in \mathbb{N}$,

$$\Pr_{(\mathbf{g}, g') \sim G(1^k), (\mathbf{a}, s) \leftarrow Z_{2^k} \times Z_{2^k}} \left[ (v, v') \leftarrow A(g^v, g'^{s}, \text{aux}_A), v' = v^a : (x, \tilde{x}) \leftarrow E_A(g^x, g'^{\tilde{x}}, \text{aux}_E) \wedge g^{(x, \tilde{x})} \neq v \right] \leq \text{negl}(k).$$

Below, we define the class of affine functions.

**Definition 2.5.** (The function family $\mathcal{F}_{aff}$). For any set $\mathcal{M}$ and any $t \in \mathbb{N}^+$, we define the following function class

$$\mathcal{F}_{aff} = \{ f(s) = d \cdot s + b \mid b, s \in \mathcal{M}, d \in \mathcal{M} \}.$$

Next we recall the definition of extractable hash of [10]. The definition can be modified to have different auxiliary inputs for adversary and extractor as the $t$-KEA above.

**Definition 2.6.** (Extractable hash [10]). An efficiently sampleable hash function ensemble $H = \{H_k\}_{k \in \mathbb{N}}$ is extractable, if for any PPT algorithm $A$, there exists a PPT extractor $E_A^k$, such that for all large $k \in \mathbb{N}$ and any auxiliary input aux$_E \in \{0, 1\}^{poly(k)}$:

$$\Pr_{h \sim H_k} \left[ y \leftarrow A(h, \text{aux}), \exists x : h(x) = y : \left( x' \leftarrow E_A^k(h, \text{aux}) \wedge h(x') \neq y \right) \right] \leq \text{negl}(k).$$

Below, we define the split-state functions class, $\mathcal{F}_{ss}$, and the $\lambda$-bit leakage function class $\mathcal{L}_\lambda$. A definition for split-state leakage functions was considered in [40].

**Definition 2.7.** (The split-state function family $\mathcal{F}_{ss}$). For any, even, $\nu \in \mathbb{N}$ and any efficiently computable function $f : \{0, 1\}^\nu \rightarrow \{0, 1\}^\nu$, $f \in \mathcal{F}_{ss}$, if there exist efficiently computable functions $f_1 : \{0, 1\}^{\nu/2} \rightarrow \{0, 1\}^{\nu/2}$, $f_2 : \{0, 1\}^{\nu/2} \rightarrow \{0, 1\}^{\nu/2}$, such that for every $x_1, x_2 \in \{0, 1\}^{\nu/2} \times \{0, 1\}^{\nu/2}$, $f(x_1 | x_2) = f_1(x_1) || f_2(x_2)$.

**Definition 2.8.** (The $\lambda$-bit leakage function class $\mathcal{L}_\lambda$). For any $\lambda \in \mathbb{N}$, $\mathcal{L}_\lambda$ is the set of the efficiently computable functions that output $\lambda$ bits, i.e., for any $g \in \mathcal{L}_\lambda$, $g : \{0, 1\}^\nu \rightarrow \{0, 1\}^\lambda$.

Next, we state the definition of semantically secure authenticated encryption, against one time leakage.

**Definition 2.9.** (Sema-tically secure authenticated encryption against one time leakage) Let $k$ be the se-curity parameter, let $\{KGen, E, D\}$ be a symmetric encryption scheme and let $\mathcal{L}$ be a set of functions. Then, $(\text{KGen}, \text{E}, \text{D})$ is authenticated, semantically secure against one-time leakage with respect to $\mathcal{L}$, if

1. (Correctness): For every message $s$, $\Pr[D_{\mathcal{D}}(E_d(s)) = s] = 1$, where $sk \leftarrow \text{KGen}(1^k)$.  

1321
2. (Semantic security): for any function $g \in \mathcal{L}$ and any two messages $s_0, s_1$, the following distributions are (either computationally or statistically) indistinguishable:

$$\left( E_{sk}(s_0), g(sk) \right) \approx \left( E_{sk}(s_1), g(sk) \right),$$

where $sk \leftarrow KGen(1^k)$.

3. (Unforgeability): For every PPT algorithm $A = (A_1, A_2)$, the probability

$$\Pr \left[ e' \neq e \land \mathcal{D}_A(e') \neq \bot \mid sk \leftarrow KGen(1^k); (s, st) \leftarrow A_1(1^k); e \leftarrow E_{sk}(s); e' \leftarrow A_2(e, st) \right]$$

is negligible in $k$.

Here, it should be noted that the leakage function is being defined by the attacker before receiving the challenge ciphertext, otherwise semantic security breaks.

3. $\ell$-MORE EXTRACTABLE HASH FUNCTION FAMILIES

In this section we define the notion of $\ell$-more extractable hash function families, and we provide a general discussion on the primitive.

**Definition 3.1. ($\ell$-more extractable hash function families) For $\ell \in \mathbb{N}$, an efficiently sampleable hash function ensemble $\mathcal{H} = \{H_k\}_{k \in \mathbb{N}}$, is $\ell$-more extractable, if for any PPT algorithm $A_0$ and any $aux_{A_0} \in \{0, 1\}^{poly(k)}$, there exist a PPT extractor $\mathcal{E}_{A_0}^N$ and $aux \in \{0, 1\}^{poly(k)}$, such that for all PPT algorithms $A_i$, any large $k \in \mathbb{N}$ and any vector of messages $s = (s_1, \ldots, s_\ell)$,

$$\Pr \left[ \mathcal{E}_{A_0, A_i, E_{A_0}^N}(\ell, aux_{A_0}, aux) = 1 \right] \leq \text{negl}(k),$$

where,

$$\mathcal{E}_{A_0, A_i, E_{A_0}^N}(\ell, aux_{A_0}, aux) :$$

$$\forall i \in [\ell], s_i \leftarrow U_{\{0, 1\}^{poly(k)}}, v_i = h_2(s_i, s)$$

(hash computation)

$$(\hat{v}, st) \leftarrow A_i(h_2, s, aux_{A_i})$$

(ham tampering)

$$(\hat{\bar{v}}, \bar{s}) \leftarrow \mathcal{E}_{A_0}^N(h_2, s, aux_{A_0})$$

(pre-image extraction)

$$h_2(\hat{\bar{s}}, \bar{s}) \neq \hat{v}, return 1$$

otherwise, return 0

The main steps in the above experiment are the following. Initially, we sample randomness for the hash, and perform the hash computation over $\ell \in \mathbb{N}$, pre-images. For deterministic hash function families we just omit randomness sampling, and we compute the hash, only using the messages. The challenge for the attacker $A_i$, is to produce a valid hash value $\bar{v}$, given $h$ has values, denoted by $v$, and auxiliary information $aux_{A_i}$. Then, the extractor $\mathcal{E}_{A_0}^N$ is executed, given $v$ and its own auxiliary input aux. Notice, that, we allow the auxiliary input of the extractor to depend on the attacker’s auxiliary input. Finally, the adversary $A_i$ produces a valid pre-image for $\bar{v}$, while given all information during the execution. The output of the experiment is 1, if $A_i$ produces a valid hash value $\bar{v}$, $A_i$ produces a valid pre-image for $\bar{v}$, while the extractor fails.

Leaving aside the fact that the above definition considers randomized function families, the major difference between the current definition and the one given by Bitansky et al. [9, 10] (Definition 2.6), is two-fold: first the “$\ell$-more” generalization that allows the attacker to have access to $\ell$ valid hash values for which it does not know the pre-images, prior to delivering its own hash value. Second, the introduction of the algorithm $A_i$, that takes the place of the existential quantifier that appears in the original definition. This is in fact a weakening of the original definition, in the sense that the extractor is allowed to fail in case a pre-image exists but is not efficiently computable based on the view of the adversary (this would not be allowed in the original definition).

Note, that, the existence of $A_i$ does not trivialize the problem for the extractor since the extractor is challenged to produce a valid pre-image for $\bar{v}$, given only the code of $A_i$, and its own auxiliary input (and in particular it lacks access to the state of $A_i$ and the program of $A_i$).

It is easy to see that, constructing $\ell$-more extractable hash function families that are non-compressing, can be achieved using existing tools, such as robust NIZKs [26]. Here we construct an $\ell$-more extractable, collision resistant, hash (ECRH) function family, achieving length-efficiency comparable to that of a regular hash function.

In the following lemma we prove that, for any $\ell$-more ECRH function family, the output of the extractor should match the output of $A_i$, in case both of them output valid pre-images, otherwise we break collision resistance.

**Lemma 3.2.** Let $\mathcal{H} = \{H_k\}_{k \in \mathbb{N}}$ be a collision resistant, $\ell$-more extractable, efficiently sampleable, hash function ensemble. Then, for any $A_i, aux_{A_i}, \mathcal{E}_{A_0}^N, aux, A_i, s = (s_1, \ldots, s_\ell), \ell$, as they were defined in Definition 3.1, the probability

$$\Pr \left[ h_2(\hat{\bar{s}}, \bar{s}) \neq \hat{v}, \hat{v} \neq v_i, i \in [\ell] \right]$$

is negligible in $k$.

For the proof see the full version of the paper.
**Lemma 3.4.** (Collision Resistance for Construction 3.3 [10]) Assuming the hardness of the discrete logarithm problem, with respect to a group $G$, Construction 3.3 is collision resistant, with respect to $G$.

The above construction is also extendable with respect to Definition 2.6 and 0-more extractable, where both properties follow from the k-KEA assumption. In the following lemma we prove that Construction 3.3 is not 1-more extractable.

**Lemma 3.5.** (Construction 3.3 is not 1-more extractable) Let $H^*$ be the hash function family of Construction 3.3, with respect to a group generation algorithm $G$. Then, assuming the difficulty of the discrete logarithm problem for $G$, $H^*$ is not 1-more extractable.

In the introduction, we argued that the hash function family of Construction 3.3 is highly malleable, and thus not 1-more extractable, under the discrete logarithm assumption. We formalize the proof in the full version.

### 4. A NON-MALLEABLE CODE AGAINST SPLIT-STATE TAMPERING

In this section, we present our construction of non-malleable codes against split-state tampering functions. Our construction requires (i) a one-time, authenticated, symmetric-key encryption scheme that is also leakage resilient, and (ii) a 1-more ECRH.

**Construction 4.1.** Let $H_k = (\text{Gen}, h)$ be a hash function family, and let $(K\text{Gen}, E, D)$ be a symmetric encryption scheme. We define a coding scheme $(\text{Init}, \text{Enc}, \text{Dec})$, as follows:

- $\text{Init}(1^k)$: sample $z \leftarrow \text{Gen}(1^k)$ and set $\Sigma = z$.
- $\text{Enc}(\Sigma, \cdot)$: let $s$ be the input to the encoder. The encoder samples $sk \leftarrow K\text{Gen}(1^k)$, $r \leftarrow \{0, 1\}^{\text{poly}(k)}$, $e \leftarrow E_k(s)$, and outputs $(r, sk, e, h_z(r, sk))$. In particular, the left part of the codeword is $(r, sk)$, while the right part is $(e, h_z(r, sk))$.
- $\text{Dec}(\Sigma, \cdot)$: let $(r, sk, e, v)$ be the input to $\text{Dec}$. If $h_z(r, sk) = v$, the decoder outputs $D_k(e)$, otherwise, it outputs $\perp$.

Since the input message to $h_z$, $sk$, possesses adequate entropy, it is possible to omit $r$ in the above construction, still for the sake of clarity we stick to the formulation provided in Definition 3.1 and we use independent randomness for hashing $sk$. In what follows we prove that Construction 4.1 is strongly non-malleable against $F_{\text{ss}}$ (Definition 2.7), assuming that for any $f = (f_1, f_2) \in F_{\text{ss}}$, $f_1$, $f_2$ affect $(r, sk)$ and $(e, v)$, independently, i.e., we assume the strings $r||sk, e||v$ are of length $\nu/2$, where $\nu$ is the length of the codeword.

**Intuition for the construction.** Before formally analyzing the construction, we first discuss the ideas on why our construction is secure. Consider a split-state tampering function $(f_1, f_2)$, where $f_1$ is applied to $(r, sk)$, and $f_2$ is applied to $(e, v)$. To prove non-malleability, roughly speaking, we need to simulate the tampering experiment without knowing the underlying message distribution. A first idea is to simulate the left side with $(r', sk')$, and the right side with $(e' = E_k(0), v' = h(r', sk'))$, where $r', sk'$ is fresh randomness and key, respectively, hoping to infer the final outcome of the tampering experiment correctly due to the semantic security of the encryption.

There are several subtleties in doing so. First and foremost, the simulator needs to be able to produce the decoding of the codeword in case $v'$ is modified by $f_2$. This is where 1-more extractability will be used to obtain a valid pre-image, $(\hat{r}, \hat{sk})$. It might be very tempting to conclude the simulation by outputting the decrypted message $D_k(\hat{e})$ (where $\hat{e}$ is the modified codeword). However, this may not be consistent with the real-world experiment, as the values produced by the extractor $(\hat{r}, \hat{sk})$ might not be consistent with the output of $f_1$. To check consistency, the simulator would want to check the output of $f_1$, yet such a simulation would be impossible to prove since it depends on $sk$, where the indistinguishability between $e'$ and $e$ does not hold in the presence of it. To go around this, we use a similar technique to Liu and Lysyanskaya [40], who observed that, the equality test between $f_1(\hat{r}, \hat{sk})$ and $f_1(r, sk)$ can be performed via the leakage of a universal hash (cf. the full version) with $\log^2 k$ bits of output. Putting this to our setting, by requiring the encryption $(K\text{Gen}, E, D)$ to be a one-time semantically secure, symmetric-key authenticated encryption, that is secure under $2k + \log^2 k$ bits of leakage, is sufficient to facilitate the simulation. We also note that the case when $v'$ is not modified by $f_2$ can be easily taken care of by the security of the authenticated encryption: as long as the key is not modified, any attempt to modify the ciphertext will result in an invalid ciphertext.

**Theorem 4.2.** Let $k$ be the security parameter, $H_k$ be a 1-more extractable hash function family that outputs $\beta(k)$ bits, $\beta(k) = \text{poly}(k)$, and let $(K\text{Gen}, E, D)$ be an authenticated, semantically secure, symmetric encryption scheme, that is leakage resilient against $L_{\text{ss}}$, $\lambda(k) = \omega(\log k) + \beta(k)$. Then, Construction 4.1 is strongly non-malleable against $F_{\text{ss}}$.

**Proof.** Following the definition of strong non-malleability (Definition 2.3), we need to prove that for any $f = (f_1, f_2) \in F_{\text{ss}}$ and any pair of messages $s_0, s_1$, $(\Sigma, \text{Tamper}_{s_0}) \simeq_c (\Sigma, \text{Tamper}_{s_1})$, where $\Sigma \leftarrow \text{Init}(1^k)$. We introduce a series of hybrids (see Figure 1), and the proof can be derived directly from the indistinguishability between adjacent hybrids. We first explain the hybrids and define the notation used in those experiments.

- Given a tampering function $f = (f_1, f_2)$ and message $s$, the first experiment, $\text{Exp}_{f, \Sigma, s}^{0}$, is exactly the original tampering game, $\text{Tamper}_{s_0}$, of Definition 2.3.
- In $\text{Exp}_{f, \Sigma, s}^{1}$, we slightly modify the previous hybrid by checking whether the function $f_2$ has modified the hash value $v$. Intuitively, by the collision resistance property of the hash function family $H_k$, if $f_2$ does not modify $v$, then the attack produces a valid codeword, $\hat{e}$, only if the parts of $\hat{e}$ that constitute the pre-image of $\hat{e}$, are kept intact, i.e., $(r, sk) = (\hat{r}, \hat{sk})$, otherwise there is a collision. In addition, assuming $sk = sk$, we have that, if $\hat{e} \neq e$, then the output of the decoder should be $\perp$, otherwise we break the authenticity under leakage $(v)$ is considered as leakage over $sk$ property of the
encryption scheme. On the other hand, if \( v \neq \bar{v} \), the output of the current experiment is produced as in \( \text{Exp}_0^{f_{\Sigma,s}} \).

- In \( \text{Exp}_2^{f_{\Sigma,s}} \), we modify the previous experiment for the case in which \( v \) is modified: instead of using the real decoding procedure, we use the extractor of the hash function family, to extract a pre-image \((r, sk)\), for \( \bar{v} \), and then compute the output, \( \hat{s} \), with respect to that pre-image. However, we cannot output \( \hat{s} \) directly as we still need to check consistency with the output of \( f \), i.e., we need to check whether \((\hat{r}, sk)\) is equal to \((\hat{r}, sk)\). The indistinguishability between the current hybrid and the previous one, follows by the 1-more extractability property of the hash function, which, informally, guarantees that if \( \hat{c} \) is a valid codeword, then \( E_{\lambda}^H \) produces a valid pre-image for \( \hat{v} \), with overwhelming probability. If the extracted pre-image is consistent with the one output by \( f \), the current hybrid outputs a non-bottom value, equal to the one output by the decoding procedure of \( \text{Exp}_0^{f_{\Sigma,s}} \). On the other hand, if \((\hat{r}, sk) \neq (\hat{r}, sk)\), Lemma 3.2 guarantees that \((r, sk)\) is a valid pre-image for \( \hat{v} \), with overwhelming probability, and the current experiment properly outputs \( 1 \). Finally, it is straightforward to see, that if \( \hat{v} \) is invalid, both experiments output \( 1 \).

In order to define the extractor \( E_{\lambda}^H \), introduced in \( \text{Exp}_2^{f_{\Sigma,s}} \), we first need to define \( A_v, aux_{A_v} \), with respect to \( h_z, v, e \), and \( f = (f_1, f_2) \). Formally, we define the following:

1. (Define \( A_v \)): \( A_v(h_z, v, aux_{A_v}) := ((f_2(aux_{A_v}, v))_{[2]}, st) \), where \( st = (f_2(aux_{A_v}, v), aux_{A_v}, v) \).
2. (Choose auxiliary info for \( A_v \)): set \( aux_{A_v} = e \).
3. (Existence of the extractor, \( E_{\lambda}^H \), and auxiliary input, \( aux_E \)): Given \( A_v \) and \( aux_{A_v} \), by the 1-more extractability property of \( H_A \), there exists an extractor \( E_{\lambda}^H \), with hardened auxiliary info, \( aux_E \), that computes \((\hat{r}, sk) \leftarrow E_{\lambda}^H(h_z, v) \).

The extractor \( E_{\lambda}^H \) is used in \( \text{Exp}_2^{f_{\Sigma,s}} \) and all subsequent experiments (for brevity we denote it as \( E \)).

We remind, that, for any vector \( v, [v]_i \), denotes the \( i \)-th coordinate of \( v \).

- In \( \text{Exp}_3^{f_{\Sigma,s}} \), we modify the consistency check procedure, so that we access the right part of the codeword, only through leakage. Instead of checking consistency using directly the output of \( f_1 \), we do the check using a random hash function, \( \hat{h} \), from a universal family (cf. full version), applied to the output of \( f_1 \), plus one more bit, that indicates whether \( f_1 \) has modified its input. Here, the hash \( v \) is computed through leakage over \( sk \). The experiment differs from the previous one only when there is a collision against \( \hat{h} \), which happens with negligible probability, as \( \hat{h} \) is a universal hash function. Below, we formalize the above procedure: let \( \hat{h} \leftarrow H_{\lambda-1} \) be a random hash function from a universal hash function family, that outputs \( \lambda - 1 \) bits. We define the function \( g_{\hat{h}, h_z}(v, y) \) as follows:

\[
g_{\hat{h}, h_z}(x, y) = \begin{cases} (0, \hat{h}(f_1(x, y)), h_z(x, y)), & \text{if } f_1(x, y) = (x, y), \\ (1, \hat{h}(f_1(x, y)), h_z(x, y)), & \text{if } f_1(x, y) \neq (x, y). \end{cases}
\]

We view \( g_{\hat{h}, h_z} \) as a leakage function that outputs \( \lambda = \omega(\log k) + \beta(k) \) bits in total. The experiment will then use the leaked value to check consistency, instead of using the whole string output by \( f_1 \). Concretely, we introduce the random variable \( b \), which depends on the output of the leakage function, and we modify \( \text{Exp}_3^{f_{\Sigma,s}} \), so that the condition “\( (b = 1) \)”, introduced in \( \text{Exp}_2^{f_{\Sigma,s}} \), is exactly the same as the condition “\( (r, sk, e) = (\hat{r}, sk, \tilde{e}) \)”, of experiment \( \text{Exp}_2^{f_{\Sigma,s}} \). This modification does not induce any statistical difference. In the next modification, we check equality between \((\hat{r}, sk)\), \((\hat{r}, sk)\), by checking if \( \hat{h}(\hat{r}, sk) = \hat{h}(f_1(\hat{r}, sk)) \). Clearly, this part induces a statistical difference only if there is a collision against \( \hat{h} \), which happens with negligible probability, since \( \hat{h} \) is a universal hash function, chosen by the current experiment, independently.

- Finally, we are going to show that \( \text{Exp}_3^{f_{\Sigma,s}} \) is indistinguishable from \( \text{Exp}_3^{f_{\Sigma,\tilde{o}}} \), for any message \( s \), where \( \tilde{o} \) denotes the zero-message. This follows by the semantic security of the leakage resilient encryption scheme (Definition 2.9).

A concrete presentation of the hybrids, is given in Figure 1. The indistinguishability between the hybrids in proved in the full version of the paper.

**Length of the CRS.** The length of the CRS in our construction is roughly \( 32k \) bits: we need to hash a \( 6k \)-bit (roughly) key of an authenticated encryption scheme and then encrypt the message using that key; this would require the parameters for the 16-KEA to be on the CRS, resulting in the \( 32k \) bits length.

5. **CONSTRUCTING 1-MORE ECRH**

In this section, we present our construction of 1-more extractable hash function families. Our construction is in two steps: (1) we first present a construction assuming a coding scheme that satisfies randomness simulatable non-malleability (RSS-NM), against affine tampering functions, and (2) we show how to construct such a code. Finally, we present Corollary 5.7 to summarize our overall construction, by putting all things together in a single statement. As we have already discussed in the introduction, the idea of constructing a NM-code for affine functions, as an intermediate step for providing split-state codes, was also followed by [4], still, our technique differs significantly, and their code does not directly satisfy our requirements. Moreover, in [22] the authors construct AMD codes, still their notions are slightly different and do fit in our framework.

### 5.1 1-more extractable hash functions from RSS-NM codes against affine functions

In this section we construct a collision resistant, 1-more extractable hash function family. Before doing so, we present the notion of “randomness simulatable, strongly non-malleable codes” (RSS-NMC). This notion is stronger than strong non-malleability in the sense that besides simulating the pre-image, \( \tilde{s} \), of the tampered codeword, the simulator also needs to produce the randomness of the decoder, \( \tilde{s}_e \), such that the encoding of \( \tilde{s} \) with randomness \( \tilde{s}_e \), produces the tampered codeword. To ease the presentation of RSS-NMC, we modify the syntax of non-malleable codes, so that the Dec algorithm returns, not only the decoded message \( \tilde{s} \), but also the
randomness string \( \tilde{s} \) for the encoder \( \text{Enc} \). This is the string that in the tampering experiment the simulator should be able to match.\(^5\)

**Definition 5.1. (Randomness simulatable, strongly non-malleable code)** Let \((\text{Enc}, \text{Dec})\) be a \((k, \nu)\)-coding scheme and \( \mathcal{F} \) be a family of functions \( f : \{0, 1\}^k \to \{0, 1\}^\nu \). For every \( f \in \mathcal{F} \) and \( z \in \{0, 1\}^\nu \), define the tampering experiment:

\[
\text{Temper}_{f, z} \overset{\text{def}}{=} \begin{cases} 
(\tilde{c}, \tilde{\tilde{c}}) & \text{sample \( \tilde{c} \leftarrow \text{Enc}(s), \tilde{\tilde{c}} \leftarrow f(c), (\tilde{s}, \tilde{\tilde{s}}) = \text{Dec}(\tilde{c}) \)} \\
\text{Output same}^* & \text{if \( \tilde{c} = c \), and (\tilde{s}, \tilde{\tilde{s}}) \) otherwise. 
\end{cases}
\]

which is a random variable over the randomness of \( \text{Enc} \) and \( \text{Dec} \). A coding scheme \((\text{Enc}, \text{Dec})\) is randomness simulatable, strongly non-malleable (RSS-NM), with respect to the function family \( \mathcal{F} \), if for every \( f \in \mathcal{F} \) and any \( z_0, z_1 \in \{0, 1\}^\nu \), we have:

\[
\text{Temper}_{f, z_0} \approx \text{Temper}_{f, z_1}
\]

where \( \approx \) may refer to statistical, or computational, indistinguishability. For coding schemes in the common reference string model, the definition is analogous.

Next we present our construction:

**Construction 5.2 (1-more extractable hash).** Let \( \mathcal{G} \) be a group-generation algorithm and let \((\text{Enc}, \text{Dec})\) be a \((kt, kt')\)-coding scheme, \( t, t' = O(\text{poly}(k)) \). An instance of a \((kt, 2kt)\)-compressing hash function family \( \mathcal{H} = (\text{Gen}, h) \) is defined as follows:

\[
\begin{align*}
\text{Exp}_{\text{Enc}, \mathcal{G}, \text{Dec}}^{\text{Tamper}}: \\
(r, sk, c, v) &\leftarrow \text{Enc}(s), (\tilde{c}, \tilde{\tilde{c}}) = (r, \tilde{c}, v) \\
\tilde{\tilde{c}} &\leftarrow f(c, \tilde{\tilde{c}}), \tilde{\tilde{c}} = \text{Dec}(\tilde{c}) \\
\tilde{\tilde{c}} &\leftarrow f(c, \tilde{\tilde{c}}), \tilde{\tilde{c}} = \text{Dec}(\tilde{c}) \\
\text{Output} &\text{same}^* \text{ if } \tilde{c} = c, \text{ and } \tilde{\tilde{c}} \text{ otherwise.}
\end{align*}
\]

1. \( \text{Gen}(1^k) \): sample \( (G, g, p) \leftarrow \mathcal{G}(1^k), (a, r) \leftarrow \mathbb{Z}_p \times \mathbb{Z}_p^* \), where \( p = |G| \), and output \( z = (G, g^a, g^b) \).

2. **Hashing computation**: on input \( s = (s_1, \ldots, s_{t}) \), sample \( \bar{u} \leftarrow U_{\{0, 1\}^{|\log(k)|}} \), compute \( h_z(s, \bar{u}) = (g^{(r, c)}, \tilde{\tilde{c}^*}) \), where \( c \leftarrow \text{Enc}(s, \bar{u}) \).

In the following we prove that Construction 5.2, which is a composition of a coding scheme \((\text{Enc}, \text{Dec})\), with construction 3.3 (the 0-more extractable hash function by Bitansky et al. [10]), is collision resistant, 1-more extractable, and uniform under leakage, assuming that \((\text{Enc}, \text{Dec})\) satisfies certain properties. Then, in Section 5.2, we instantiate \((\text{Enc}, \text{Dec})\) with the desired properties. Below, we prove that Construction 5.2 is collision resistant.

**Lemma 5.3.** Let \( \mathcal{G} \) be any group generation algorithm. Then, assuming the hardness of the discrete logarithm problem on \( \mathcal{G} \), and the underlying encoding algorithm is injective, Construction 5.2 is collision resistant with respect to \( \mathcal{G} \).

**Proof.** The work [10] proves that the hash function family of Construction 3.3, i.e., \( \mathcal{H}^* \), is collision resistant, assuming the difficulty of the discrete logarithm problem. We note that Construction 5.2 is a composition of \( \text{Enc}() \) and \( \mathcal{H}^* \). Following a simple fact that any injective function composed with a collision resistant hash function still results in a collision resistant hash function (composition in any order), we can conclude that the hash function family of Construction 5.2 is collision resistant, under the same assumption.

In the following theorem, we prove that, under certain assumptions, Construction 5.2, is 1-more extractable.
Theorem 5.4. Let \( t(k), t'(k) = O(\text{poly}(k)) \), (Enc, Dec) be any RSS-non-malleable, \((kt, kt')\)-coding scheme, against \( \mathcal{F}_{\text{aff}} \), let \( \mathcal{H} \) be the hash function family of Construction 5.2 with respect to \((\text{Init}, \text{Enc}, \text{Dec})\), and assume that for any message \( s \), \( H_{\infty}(\text{Enc}(s)) \geq k + \omega(\log k) \). Then, assuming \( t'\)-KEA and the hardness of DLog, \( \mathcal{H} \) is 1-more extractable, with respect to \((\text{Init}, \text{Enc}, \text{Dec})\).

Proof. For \( k \in \mathbb{N} \), let \((\text{Enc}, \text{Dec})\) be an RSS-NM, \((kt, kt')\)-coding scheme, against \( \mathcal{F}_{\text{aff}} \), let \( \mathcal{H} \) be the \((kt, 2k)\)-compressing, collision-resistant, hash function family of Construction 5.2. Following Definition 3.1, we need to prove that for any PPT algorithm \( \mathcal{A}_v \) with auxiliary input \( \text{aux}_{\mathcal{A}_v} \), there exist extractor \( \mathcal{E}_{\mathcal{A}_v}^H \), and auxiliary input \( \text{aux}_e \), such that for any PPT algorithm \( \mathcal{A}_v \), any large \( k \) and every message \( s = (s_1, \ldots, s_t) \in \mathbb{Z}_p^t \),

\[
\Pr_{h_z \leftarrow \mathcal{H}} \left[ \text{Exp}_{\mathcal{A}_v, \mathcal{A}_s, \mathcal{E}_{\mathcal{A}_v}^H}(1, \text{aux}_{\mathcal{A}_v}, \text{aux}_e) = 1 \right] \leq \text{negl}(k). \tag{1}
\]

Clearly, if \( \mathcal{A}_v \) fails to produce a new valid hash, or, if \( \mathcal{A}_v \) fails to produce a valid pre-image for the new hash, the experiment simply outputs 0, and there is no challenge for the extractor. Therefore, the interesting case is when \( \mathcal{A}_v \) produces a valid hash value, say \( \hat{v} \), while having access to an element in the range of the hash, say \( v \), and \( \mathcal{A}_v \) produces a valid pre-image for \( \hat{v} \), while having access to \( s, v, \hat{v} \), and any other state information produced by \( \mathcal{A}_v \). Hence, for the rest of the proof we assume \( \hat{v} \neq v \), and \((\hat{s}, \hat{s})\) is a valid pre-image for \( \hat{v} \), i.e., \( h_z(\hat{s}, \hat{s}) = \hat{v} \).

Given \( \mathcal{A}_v \) with auxiliary input \( \text{aux}_{\mathcal{A}_v} \), the idea behind the definition of the extractor, \( \mathcal{E}_{\mathcal{A}_v}^H \), and its auxiliary input, \( \text{aux}_e \), goes as follows:

- First we define an adversary against the hash function family \( \mathcal{H}^* \), of Construction 3.3: \( \mathcal{A}_v(h_z^*, \text{aux}_{\mathcal{A}_v}) := \mathcal{A}_v(h_z, v, \text{aux}_{\mathcal{A}_v}) \), where \( \mathcal{A}_v \) first interprets the description of the hash function \( h_z^* \), as \( (h_z, v) \), i.e., as a description of a hash function in \( \mathcal{H} \) and a hash value \( v \), and then executes \( \mathcal{A}_v(h_z, v, \text{aux}_{\mathcal{A}_v}) \). The function \( h_z^* \) will be stated concretely below.

- Since \( \mathcal{H}^* \) is a 0-more extractable hash function family, and assuming \( h_z^* \) is indistinguishable from an element in \( \mathcal{H}^* \), there exists an extractor \( \mathcal{E}_{\mathcal{A}_v}^{H^*} \) with its auxiliary input \( \text{aux}_e \), that extracts a valid pre-image for \( \hat{v} \), with respect to \( h_z^* \). We define the auxiliary input \( \text{aux}_e := \text{aux}_e \).

The extractor is defined below.

The extractor \( \mathcal{E}_{\mathcal{A}_v}^{H^*} \):

Input: \( z = (g^t, g^{tv}), v = (g^{tv}, g^{tv'}) \), \( \text{aux}_e \).

1. Set \( z' = (g^t, g^{tv}, g^{tv'}, g^{tv''}) \). Here, we interpret \( z' \) as a description of hash function \( h_z^* \) for vector messages with \( t'+1 \) coordinates.
2. Sample \((b_1, \ldots, b_t', d) \leftarrow \mathcal{E}_{\mathcal{A}_v}^{H^*}(h_z^*, \text{aux}_e) \) and set \( f = (b_1, \ldots, b_t', d) \in \mathbb{Z}_p^{t'+1} \).
3. Interpret \( f \) as an affine function that on input \((x_1, \ldots, x_{t'}) \) outputs \((dx_1 + b_1, dx_2 + b_2, \ldots, dx_t + b_t') \), and then sample \((\bar{s}, \bar{s}) \leftarrow \mathcal{D}_{\mathcal{A}_v}^H \), where \( \mathcal{D}_{\mathcal{A}_v}^H \) is the simulator of the underlying RSS-NM code, \( (\text{Enc}, \text{Dec}) \), parameterized by the affine function \( f \).
4. Output: \((s, s)\).

The extractor is defined with respect to any input \( v \), still by the definition of the \( t' \)-more experiment, \( v \) is always a valid hash value, i.e., \( v = h_z(s) = (g^{tv}, g^{tv'}) \), where \( c \leftarrow \text{Enc}(s, s) \), for some message \( s \). Then, for any \( \mathcal{A}_v \), and message \( s \), we are going to analyze the execution of \( \text{Exp}_{\mathcal{A}_v, \mathcal{A}_s, \mathcal{E}_{\mathcal{A}_v}^{H^*}}(1, \text{aux}_{\mathcal{A}_v}, \text{aux}_e) \). We first prove that with overwhelming probability, the following events happen:

- \( E_1: h_z^*(b_1, \ldots, b_t', d) = \hat{v} \). Recall that \( \hat{v} \) is the output of \( \mathcal{A}_v \) on input \((h_z, v)\).
- \( E_2: \text{Enc}(\bar{s}, a) = f(c) \). Recall that \((\hat{s}, \hat{s})\) is the output of \( \mathcal{A}_v \).

We formalize those ideas in a series of claims, where their concrete proofs are given in the full version of the paper.

5.2 Constructing RSS-NM codes

In this section, we construct RSS-NM codes as required by the previous section.

Construction 5.5 (The code for \( \mathcal{F}_{\text{aff}} \)). For any \( k \in \mathbb{N}, t = O(\text{poly}(k)) \), we define a \((kt, (2t + 4)k)\)-coding scheme \((\text{Init}, \text{Enc}, \text{Dec})\) in the CRS model\(^9\) as follows:

- \( \text{Init}(1^k) \): sample a \( k \)-bit prime \( p \in (2^{k-1}, 2^k) \) and set \( \Sigma = p \).
- \( \text{Enc}(\Sigma, \cdot) \): let \( s = (s_1, \ldots, s_t) \in \mathbb{F}_p^t \) be the input to \( \text{Enc} \). Sample two random field elements \( v, r \in \mathbb{F}_p \) and then output \( c = (v, v^2, r, r^2, u_1, u_2, \ldots, u_{t+1}^2) \in \mathbb{F}_p^{2t+4} \), where \( u_i = s_i - r \) for \( i \in [t] \).
- \( \text{Dec}(\Sigma, \cdot) \): on input \( c = (v, v^2, r, r^2, u_1, u_2, \ldots, u_{t+1}^2) \), the decoder checks whether \( v = v^2, r = r^2 \), and \( u_i = u_i^2 \) for all \( i \in [t] \). If so, then it outputs \((v, r, u_1 + r, u_2 + r, \ldots, u_{t+1}) \), otherwise, outputs \( \bot \).

All operations are performed modulo \( p \). We also consider the deterministic version of \( \text{Enc} \) by allowing the randomness to be given on the input. In that case we have \( c = \text{Enc}(\Sigma, s, s) \), where \( s = (v, r) \).

Notice, that, the randomness employed by the above construction is \( 2k \), independently of the message length.

Theorem 5.6. The code of Construction 5.5 is randomness simulatable, strongly non-malleable (Definition 5.1), with respect to \( \mathcal{F}_{\text{aff}} \). In addition, for any message \( s, H_{\infty}(\text{Enc}(s)) \geq k + \omega(\log k) \).

The proof intuition is presented in the introduction and the concrete proof is given in the full version of this paper.

5.3 Our resulting instantiation

By plugging Construction 5.5, as the underlying coding scheme to Construction 5.2, we receive the corollary:

Corollary 5.7. Under the DLOG assumption and \( t \)-KEA, there exists a 1-more extractable, collision resistant, hash function family \( \mathcal{H}_k \).

Proof. Let \((\text{Init}, \text{Enc}, \text{Dec})\) be the \((kt, (2t + 4)k)\), RSS-NM code of Construction 5.5. Then we construct \( \mathcal{H}_k \) by plugging in \((\text{Init}, \text{Enc}, \text{Dec})\), as the underlying coding scheme to the hash function family of Construction 5.2. Clearly,

\(^9\)Note that the CRS is not essential for this encoding, but for simplicity we describe the code in this model.
6. CONSTRUCTING ℓ-MORE EXTRACTABLE HASH

In the “ℓ-more” setting, the attacker is given $v_1, \ldots, v_\ell$, hash values, and produces a new hash value $\tilde{v}$. Having the techniques from the “1-more” setting, one can easily argue the attack against $\tilde{v}$ (in the ℓ-mo...