Locally Decodable and Updatable Non-Malleable Codes in the Bounded Retrieval Model

Dana Dachman-Soled, Mukul Kulkarni, and Aria Shahverdi

University of Maryland, College Park, USA
{danadach@ece., mukul@terpmail., ariash@}umd.edu

Abstract. In a recent result, Dachman-Soled et al. (TCC '15) proposed a new notion called locally decodable and updatable non-malleable codes, which informally, provides the security guarantees of a non-malleable code while also allowing for efficient random access. They also considered locally decodable and updatable non-malleable codes that are *leakage-resilient*, allowing for adversaries who continually leak information in addition to tampering.

The bounded retrieval model (BRM) (cf. [Alwen et al., CRYPTO '09] and [Alwen et al., EUROCRYPT '10]) has been studied extensively in the setting of leakage resilience for cryptographic primitives. This threat model assumes that an attacker can learn information about the secret key, subject only to the constraint that the overall amount of leaked information is upper bounded by some value. The goal is then to construct cryptosystems whose secret key length grows with the amount of leakage, but whose runtime (assuming random access to the secret key) is *independent* of the leakage amount.

In this work, we combine the above two notions and construct locally decodable and updatable non-malleable codes in the split-state model, that are secure against *bounded retrieval* adversaries. Specifically, given leakage parameter ℓ, we show how to construct an efficient, 3-split-state, locally decodable and updatable code (with CRS) that is secure against one-time leakage of any polynomial time, 3-split-state leakage function whose output length is at most ℓ, and one-time tampering via any polynomial-time 3-split-state tampering function. The locality we achieve is polylogarithmic in the security parameter.

1 Introduction

Non-malleable codes were introduced by Dziembowski, Pietrzak and Wichs [38] as a relaxation of error-correcting codes, and are useful in settings where privacy—but not necessarily correctness—is desired. Informally, a coding scheme is *non-malleable* against a tampering function if by tampering with the codeword, the function can either keep the underlying message unchanged or change it to an unrelated message. The main application of non-malleable codes proposed in the literature is for achieving security against leakage and tampering attacks on memory (so-called *physical attacks* or *hardware attacks*), although non-malleable codes have also found applications in other of areas of cryptography [25,24,43] and theoretical computer science [20].

In this work, we go beyond considering non-malleable codes in the context of physical and/or hardware attacks: We consider the application of a variant of non-malleable codes, known as locally decodable and updatable non-malleable codes (LDUNMC) [27], to the problem of providing data assurance in a *network* environment. Our main focus is on providing privacy and integrity for large amounts of dynamic data (such as a large medical database with many authorized users), while allowing for efficient, random access to the data. Moreover, we are interested in settings where *all* persistent data is assumed vulnerable to attack and there is no portion of memory that is assumed to be fully protected.

In the following, we provide context for the contribution of this work by first discussing the limitations of standard non-malleable codes, then discussing the recently introduced notion of LDUNMC and finally discussing why previous constructions of LDUNMC fall short in our setting.

Drawbacks of standard non-malleable codes. Standard non-malleable codes are useful for protecting small amounts of secret data stored on a device (e.g. cryptographic secret key) but unfortunately are not suitable in settings where, say, an entire database must be protected. This is due to the fact that non-malleable codes do not allow for random access: Once the database is encoded via a non-malleable code, in order to access just a single location, the entire database must first be decoded, requiring a linear scan over the database. Similarly, to update a single location, the entire database must be decoded, updated and re-encoded.

Locally decodable and updatable non-malleable codes (LDUNMC*).* In a recent result, [27] proposed a new notion called LDUNMC, which informally speaking, provides the security guarantees of a non-malleable code while also allowing for efficient random access. In more detail, we consider a database $D = D_1, \ldots, D_n$ consisting of n blocks, and an encoding algorithm $\mathsf{ENC}(D)$ that outputs a codeword $C = C_1, \ldots, C_{\hat{n}}$ consisting of $\hat{n}$ blocks. As introduced by Katz and Trevisan [51], local decodability means that in order to retrieve a single block of the underlying database, one does not need to read through the whole codeword but rather, one can access just a few locations of the codeword. In 2014, Chandran et al. [17] introduced the notion of local updatability, which means that in order to update (or "re-encode") a single block of the underlying database, one only needs to update a few blocks of the codeword.

As observed by [27], achieving these locality properties requires a modification of the definition of non-malleability: Suppose a tampering function f only modifies one block of the codeword, then it is likely that the output of the decoding algorithm, DEC, remains unchanged in most locations. (Recall DEC gets as input an index $i \in [n]$ and will only access a few blocks of the codeword to recover the i-th block of the database, so it may not detect the modification.) In this case, the (overall) decoding of the tampered codeword $f(C)$ (i.e. $(\mathsf{DEC}^{f(C)}(1), \ldots, \mathsf{DEC}^{f(C)}(n))$) can be highly related to the original message, which intuitively means it is highly malleable.

To handle this issue, [27] consider a more fine-grained experiment. Informally, they require that for any tampering function f (within some class), there exists a simulator that computes a vector of decoded messages D^* and a set of indices $\mathcal{I} \subseteq [n]$. Here $\mathcal{I}$ denotes the coordinates of the underlying messages that have been tampered with. If $\mathcal{I} = [n]$, then the simulator thinks that the decoded messages are D^*, which should be unrelated to the original messages. On the other hand, if $\mathcal{I} \subsetneq [n]$, the simulator thinks that all the messages not in $\mathcal{I}$ remain unchanged, while those in $\mathcal{I}$ become $\perp$. This intuitively means the tampering function can do only one of the following:

1. It destroys a block (or blocks) of the underlying messages while keeping the other blocks unchanged, OR
2. If it modifies a block of the underlying message to a valid encoding, then it must have modified *all* blocks to encodings of unrelated messages, thus destroying the original message.

It turns out, as shown by [27], that the above is sufficient for achieving tamper-resilience for RAM computations. Specifically, the above (together with an ORAM scheme) yields a compiler for any RAM program with the guarantee that any adversary who gets input/output access to the compiled RAM program Π running on compiled database D and can additionally apply tampering functions $f \in \mathcal{F}$ to the database D adaptively throughout the computation, learns no more than what can be learned given only input/output access to Π running on database D. Dachman-Soled et al. in [27] considered LDUNMC that are also *leakage-resilient*, thus allowing for adversaries who continually leak information about D in addition to tampering.

Drawbacks of previous constructions of LDUNMC*.* The final constructions given in [27] achieved a leakage resilient, LDUNMC in the *split-state* and *relative leakage* model. In the split-state model, the codeword C is divided into sections called split-states and it is assumed that adversarial tampering and leakage on each section is *independent*. In the relative leakage model, the amount of information the adversary can leak is at most ℓ bits, and *all parameters of the system* (including size of memory as well as complexity of $\mathsf{ENC}/\mathsf{DEC}$) scale with ℓ. Thus [27] achieves security against adversaries who continually apply arbitrary split-state tampering functions and who leak a bounded amount of information from each of the split-states in each round. A main drawback of the construction of [27] is that since their result is in the relative leakage model, the efficiency of the $\mathsf{ENC}/\mathsf{DEC}$ procedures scales with the amount of leakage ℓ allowed from one of the two split-states, which gives rise to the following dilemma: If the amount of leakage, ℓ, is allowed to be large, e.g. $\ell := \Omega(n) \cdot \log|\hat{\Sigma}|$ bits, where $\log|\hat{\Sigma}|$ is the number of bits in each block of the codeword, then *locality* is compromised, since $\mathsf{ENC}/\mathsf{DEC}$ must now have complexity that scales with $\Omega(n) \cdot \log |\hat{\Sigma}|$ and thus will need to read/write to at least $\Omega(n)$ data blocks. On the other hand, if it is required that $\mathsf{ENC}/\mathsf{DEC}$ have locality at most $\mathrm{polylog}(n)$, then it means that leakage of at most ℓ bits, where $\ell := c \cdot \mathrm{polylog}(n) \cdot \log |\hat{\Sigma}|$, for some $c < 1$, can be tolerated. In some cases—e.g. a setting in which the adversary runs a side-channel attack measuring power consumption—it may be reasonable to assume that the amount of information leaked is proportional to the complexity of $\mathsf{ENC}/\mathsf{DEC}$, but in other cases this assumption seems difficult to defend. Indeed, in this work, we allow the adversary's leakage budget to be much larger than the complexity of $\mathsf{ENC}/\mathsf{DEC}$, motivated by a network setting, in which the adversary typically corrupts a server and modifies memory while downloading large amounts of data. We do assume that if an adversary surpasses some large leakage budget, its behavior will be detected and halted by network security monitors. Thus, an adversary cannot simply download the entire encoded database (in which case security would be impossible to achieve) without being caught.

1.1 Our Results

In this work, we construct LDUNMC in the split-state model, that are secure against *bounded retrieval* adversaries. The bounded retrieval model (BRM) (cf. [9,8]) has been studied extensively in the setting of leakage resilience for cryptographic primitives (such as public key encryption, digital signatures and identification schemes). This threat model assumes that an attacker can repeatedly and adaptively learn information about the secret key, subject only to the constraint that the overall amount of leaked information is upper bounded by some value. Cryptosystems in the BRM have the property that while the secret key length grows with the amount of leakage, the runtime (assuming random access to the secret key) is *independent* of the leakage amount. In particular, the parameters of interest in a bounded retrieval model cryptosystem are the following: (1) The leakage parameter ℓ, which gives the upper bound on the overall amount of leakage; (2) The locality t of the scheme which determines the number of locations of the secret key that must be accessed to perform an operation (e.g. decryption or signing); and (3) The relative leakage $\alpha := \ell/|\mathsf{sk}|$, which gives the ratio of the amount of leakage to secret key length. Since we consider bounded retrieval adversaries in the context of locally decodable and updatable codes, our threat model differs from the standard BRM threat model in the following ways:

- We consider the protection of arbitrary data (not limited to the secret key of a cryptosystem).
- We allow adversarial tampering in addition to bounded leakage and do not assume that any portion of memory is tamper-proof[1].
- We assume that both leakage and tampering are *split-state*, i.e. that the leakage/tampering functions are applied independently to different sections of memory. Note that the *tampering* functions we consider are unrestricted (other than requiring them to be split-state and efficiently computable). The only restriction on the *leakage* functions (other than requiring them to be split-state and efficiently computable) is an upper bound on the output length (as in the BRM setting).

In our setting, we retain the same parameters of interest ℓ, t, α as above, with the only differences that each split-state must be able to tolerate at least ℓ bits of leakage and the overall relative leakage, α is taken to be the minimum relative leakage over all split-states, where the relative leakage for each split-state is computed as the maximum amount of allowed leakage for that split-state (which may be greater than ℓ) divided by the size of that split-state.

In this work, we additionally restrict ourselves to a one-time tampering and leakage model (we discuss below the difficulties of extending to a fully continuous setting), where the experiment proceeds as follows: The adversary interacts with a challenger in an arbitrary polynomial number of rounds and may adaptively choose two rounds i, j where $i \leq j$, specifying a single leakage function $g \in \mathcal{G}$ in round i and a single tampering function $f \in \mathcal{F}$ in round j. Note the adversary gets to observe the leakage in round i before specifying tampering function f in round j. At the end of the experiment, the entire decoding of the (corrupted) codeword in each round is released in addition to the leakage obtained in round i. Briefly, our security

[1] [36] also allowed for tampering in the BRM setting, but required portions of memory to be completely tamper-proof.

requirement follows the ideal/real paradigm and requires that a simulator can simulate the output of the leakage as well as the decoding of *each position in each round*, without knowing the underlying encoded message. More precisely, as in the definition of [27], in each round the simulator outputs a vector of decoded data D^* along with a set of indices $\mathcal{I} \subseteq [n]$. Here $\mathcal{I}$ denotes the coordinates of the underlying messages that have been tampered with. If $\mathcal{I} = [n]$, then the simulator must output a complete vector of decoded messages D^*. On the other hand, if $\mathcal{I} \subsetneq [n]$, the simulator gets to output "same" for all messages not in $\mathcal{I}$, while those in $\mathcal{I}$ become $\perp$. When the output of the real and ideal experiments are compared, positions designated as "same" are replaced with either the original data in that position or with the most recent value placed in that position by an update instruction.

We obtain the following result:

Theorem 1 (Informal). *Under standard assumptions we have that for security parameter $\lambda \in \mathbb{N}$ and $\ell := \ell(\lambda)$, there exists an efficient, 3-split-state,* LDUNMC *(with CRS) in the bounded retrieval model that is secure against one-time tampering and leakage for tampering class $\mathcal{F}$ and leakage class $\mathcal{G}$, where*

- *$\mathcal{F}$ consists of all efficient, 3-split-state functions $f = (f_1, f_2, f_3)$.*
- *$\mathcal{G}$ consists of all efficient, 3-split-state functions $g = (g_1, g_2, g_3)$, such that g_1, g_2, g_3 each output at most ℓ bits.*

Moreover, the scheme has locality $t := t(\lambda) \in \mathrm{polylog}(\lambda)$ and relative leakage $\alpha := \alpha(\lambda) \in \frac{1}{8} - o(1)$

Combining our notion with the results of [27] we obtain a compiler for secure RAM computation against 3-split-state adversaries in the BRM.

Applications. Our encoding scheme can be used to protect data privacy and integrity while a RAM program is being computed on the data in situations where: (1) the data is stored across at least 3 servers, (2) the attacker can corrupt all servers and launch a fully coordinated attack, (3) the attacker cannot download too much data from any of the servers at once. (3) can be justified by assuming either that the attacker has limited storage capacity or that an attacker who tries to download too much data will be detected by network security monitors. The advantage of using our approach of LDUNMC versus simply using encryption to achieve privacy and a Merkle tree to achieve integrity is that our approach allows tampering and leakage on *all* persistent data, whereas the former approach requires certain parts of memory (e.g. the parts storing the secret keys for encryption/decryption and the root of the Merkle tree) to be leak-free and tamper-free.

Difficulty of achieving continual tampering and leakage in the BRM setting. In contrast to the original definition of [27], where in each round the attacker continually specifies new f, g functions to be applied, our construction achieves a one-time notion of leakage and tamper resilience where a single f, g are applied. There seems to be an inherent difficulty in achieving continual security in the BRM model since an attacker who continually leaks may be able to simply internally simulate the *decode* algorithm and each time a read request is issued, leak the required information from the appropriate split-state, thus trivially breaking the privacy guarantee. Note that the overall leakage of such an attack *must* be bounded—and so will always qualify as a BRM attack—since the local decodability property guarantees that only a small number of locations will be read. Indeed, the construction we present in this work is vulnerable to such an attack: We store the first part of the secret key of an encryption scheme in one split-state, the second part of the secret key of an encryption scheme in a second split-state and ciphertexts in the third split-state. An attacker can first leak a target ciphertext from the third split-state, learn the locations required for decryption from the first split-state, leak those locations, then learn the locations required for decryption from the second split-state and leak those locations. Note that it is unlikely that during the course of this three-round attack any of the relevant locations in the first and second split-states are overwritten by the updater, since the few locations accessed by an update are randomly distributed. Thus, after three rounds of leakage the attacker has sufficient information to decrypt the target ciphertext. We leave open the question of whether such an attack can be generalized to show an impossibility result for continual LDUNMC in the BRM.

1.2 Techniques

Our construction proceeds in three stages:

Leakage Only. Here the attacker submits a single split-state leakage function $g := (g_1, g_2)$ and is not allowed to tamper. To achieve security, we encrypt the database block-by-block using a CPA-secure public key encryption (PKE) scheme in the BRM model. The codeword then has two split-states: The first contains the secret key and the second contains the ciphertexts. The locality and relative leakage of the scheme will be the same as that of the underlying encryption scheme. Note that even though the leakage is on both the secret key and ciphertexts, regular BRM security is sufficient for the underlying encryption scheme (and after-the-fact leakage—leakage on the secret key after seeing the ciphertext—is not necessary) since the leakage $g := (g_1, g_2)$ on both split-states is submitted simultaneously.

Leakage and Partial Tampering. Here the attacker submits a split-state leakage function $g := (g_1, g_2)$ followed by a split-state tampering function $f = (f_1, f_2)$, where f_1 is required to be the *identity function.* In terms of the previous construction, this means that the attacker gets to tamper with the ciphertexts only, but not the secret key. A first attempt to extend the construction described above is to use a CCA-secure PKE scheme in the BRM instead of a CPA-secure scheme. Indeed, this will allow the simulator to decrypt any encrypted blocks of the database that have been tampered with, thus ruling out mauling attacks on any individual block. Unfortunately, it does not prevent mauling attacks across blocks. Namely, some encrypted blocks can be replaced with fresh valid encryptions while others remain the same, leading to a valid, decoded database which is different, but correlated to the original data. To prevent this, we tie together the encrypted blocks in the database using a Merkle tree and store the Merkle tree in the second split-state along with the ciphertexts. During decode and update, we check that the relevant ciphertext block is consistent with the Merkle root. Unfortunately, this still does not work since the tampering function f_2 can be used to update the Merkle tree and root to be consistent with the modified ciphertexts at the leaves. Therefore, we additionally store a secret key for a signature scheme in the BRM model in the first split-state and include a signature on the root of the Merkle tree in the second split-state, which is verified during each decode and update. Note, however, that existentially unforgeable signatures are impossible in the BRM model, since an attacker can always use its leakage query g_1 to learn a signature on a new message. Nevertheless, so-called *entropic* signatures are possible [9,10][2]. Loosely speaking, entropic signatures guarantee unforgeability for message distributions that have high entropy, even conditioned on the adversary's view after receiving the output of g_1 but before receiving the output of g_2. Thus, to argue non-malleability we show that either (1) The ciphertexts contained in the second split-state are *all* low entropy, conditioned on the adversary's view after receiving the output of g_1 but before receiving the output of g_2. In this case, it means that $\mathcal{I} = [n]$, i.e. all the ciphertexts have been modified and so the decryption across all blocks will lead to an unrelated database or (2) At least one ciphertext has high entropy, conditioned on the adversary's view after receiving the output of g_1 but before receiving the output of g_2. In this case, we argue that the root of the Merkle tree has high entropy and so an adversary who produces a forged signature violates the entropic security of the BRM signature scheme. See Section 7.1.

Full Leakage and Tampering. When trying to extend the above construction to allow tampering on the secret (and public) keys, it becomes clear that the entire secret key cannot be stored in a single split-state due to the following trivial attack: The adversary can leak a single ciphertext from the second split-state using leakage function g_2 and subsequently tamper with the first split-state using a tampering function f_1 defined such that f_1 does nothing if the leaked ciphertext decrypts to 0, and erases the entire contents of the first split-state otherwise. Such an attack clearly breaks non-malleability, since the entire codeword will decode

[2] We note that the constructions cited above are in the random oracle model. However, as discussed in Remark 2 in Section 7.1, the primitive we require is slightly weaker than regular entropic signatures in the BRM and can be constructed in a straightforward manner in the standard model, without use of random oracles. Nevertheless, for conceptual simplicity we present our constructions and state our theorems in terms of the existence of entropic signatures in the BRM.

properly if the leaked block contained a 0, and decode to $\perp$ otherwise. To overcome this trivial attack, we introduce a third split-state: Specifically, we store the secret keys across the first two split-states and store the public keys and ciphertexts in the third. Additionally, we replace the CCA-secure public key encryption scheme in the BRM with a new primitive we introduce called *CCA secure SS-BRM public key encryption*, which may be of independent interest (See Section 6 for the definition and Section 6.1 for a construction and security proof). At a high level, given leakage parameter ℓ, such an encryption scheme stores the secret key in a split-state and guarantees CCA security even under ℓ bits of split-state leakage both before and *after* seeing the challenge ciphertext. While CCA security under after-the-fact leakage is normally impossible to achieve, we bypass this since the key is stored in a split-state. Given an encryption scheme as above, the final construction is as follows: The first split-state stores the first part of secret key of the SS-BRM PKE scheme, the secret key of the BRM signature scheme, and a Merkle tree of the former two keys with root R_1. The second split-state stores the second part of the secret key of the SS-BRM PKE scheme, and a Merkle tree of the key with root R_2. The third split-state contains the ciphertexts, Merkle tree and signature on the root as in the previous construction and, in addition, stores a simulation-sound NIZK proof of knowledge of the pre-images of R_1 and R_2, with *local* verifiability (this is the reason a CRS is necessary). Decode and update proceed as in the previous construction and additionally, during each decode and update, the proof is verified using the local verifier. Moreover, each time a location is accessed in the first or second split-state during decode or update, the corresponding locations in the Merkle tree of the first and second split-state are checked and compared with the R_1 and R_2 values contained in the proof statement in the third split-state. If they do no match, an error is outputted. In the security proof, we reduce a leakage/tampering adversary A to an adversary A' against the SS-BRM PKE scheme. To achieve this, when A submits its tampering query $f = (f_1, f_2, f_3)$, A' will use its post challenge leakage query to output a bit corresponding to each leaf block in the first and second split-state indicating whether the block is consistent with the corresponding Merkle root, R_1 or R_2. Now in order to decode and update there are two cases, if the hash values R_1 or R_2 change, then the statement of the NIZK changes and a candidate encryption and signature secret key can be extracted. If the public keys, R_1 and R_2 do not change, then the candidate encryption and signature secret keys are the original keys and the CCA oracle can be used for decryption. In addition, during each decode and update, before the candidate secret keys are used to perform the decryption or signing operation, the post-challenge leakage is used to verify that the corresponding blocks needed to perform the operation are consistent with the R_1 and R_2 values contained in the proof. If yes, the candidate key is used to decrypt or sign. If not, then an error is produced. See Section 7.2.

On 2 vs 3 split-states. The solution described above requires 3 split-states. A natural question is whether we can reduce the number of split-states to 2. This would be optimal in our setting, since if the entire codeword is stored in a single state, the adversary can request leakage corresponding to the decoding of some particular position $\mathsf{DEC}(i), i \in [n]$, thus breaking security.

Towards answering this question, recall our newly introduced notion called *CCA secure SS-BRM public key encryption*, (described in the previous section), which given leakage parameter ℓ, stores the secret key in a split-state and guarantees CCA security even under ℓ bits of split-state leakage before and *after* seeing the challenge ciphertext. This notion necessarily gives rise to a 3-split-state construction of LDUNMC in the BRM model, since each split-state of the key and the ciphertext must be stored separately. We note that our construction of CCA secure SS-BRM public key encryption, given in Section 6.3, actually achieves a stronger notion of security, where the secret key $\mathsf{sk} := \mathsf{sk}_1 || \mathsf{sk}_2$ is split into two parts and two phases of leakage are allowed: In the first phase, leakage is allowed on sk_1 and on (sk_2, c), where c is the challenge ciphertext. Then, the challenge ciphertext is given to the adversary and an additional leakage query is allowed on sk_1 and on sk_2. This notion seems useful for achieving 2-split-state construction of LDUNMC in the BRM model, since sk_1 can be stored in one split-state and (sk_2, c) in the other, this approach does not work for our construction (as we elaborate below). We therefore choose to present the simpler (and sufficient for our setting) notion of *CCA secure SS-BRM public key encryption* in Section 6.2 and construction/security proof in Section 6.3.

The above does not help in reducing our construction from 3 to 2 split-states since our construction requires one of the split-states (which contains the ciphertexts, the Merkle tree, the signature on the root and and the simulation-sound NIZK proof of knowledge) to be *entirely public*, and thus must be stored in a separate state, apart from both sk_1 and sk_2. This is done so we can fully simulate the entire contents of the split-state after the tampering function has been applied, which enables us to use the NIZK knowledge extractor to extract the encryption and signature secret keys from the (tampered) NIZK proof (as described previously). Note that we cannot rely on the BRM security of the encryption/signature scheme to allow us to instead *leak* the extracted witness, since the witness corresponds to the encryption and signature secret keys, which are required to be larger than the allowed leakage bound, ℓ, in order for security of the encryption/signature scheme to be possible.

1.3 Related Work

Non-Malleable Codes. The concept of non-malleability, introduced by Dolev, Dwork and Naor [33] has been applied widely in cryptography, in both the computational and information-theoretic setting. Error-correcting codes and early works on tamper resilience [42,47] gave rise to the study of non-malleable codes.

The notion of non-malleable codes was formalized in the seminal work of Dziembowski, Pietrzak and Wichs [38]. Split state classes of tampering functions introduced by Liu and Lysyanskaya [57], have subsequently received much attention with a sequence of improvements achieving reduced number of states, improved rate, or other desirable features [35,3,19,2,6,1,53]. Recently [11,5] gave efficient constructions of non-malleable codes for "non-compartmentalized" tampering function classes. Other works on non-malleable codes include [39,22,4,49,16,2,18]. We guide the interested reader to [50] and [57] for a discussion of various models for tamper and leakage resilience. There are also several inefficient, existential or randomized constructions for much more general classes of functions in addition to those above [38,21,41]. Choi et al. [23], in the context of designing UC secure protocols via tamperable hardware tokens, consider a variant of non-malleable codes which has *deterministic* encoding and decoding. In contrast, our work relies on both randomized encoding and decoding, as does the recent work of [11]. Chandran et al. [17] introduced the notion of locally updatable and locally decodable codes. This was extended by Dachman-Soled et al. [27] who introduced the notion of *locally decodable and updatable non-malleable codes* with the application of constructing compilers that transform any RAM machine into a RAM machine secure against leakage and tampering. This application was also studied by Faust et al. [40]. Recently, Chandran et al. [18] studied information-theoretic locally decodable and updatable non-malleable codes.

Memory Leakage Attacks. Recently, the area of *Leakage Resilient Cryptography* has received much attention by the community. Here, the goal is to design cryptographic primitives resistant to arbitrary side-channel attacks, permitting the adversary to learn information about the secret key adaptively, as long as the *total amount* of leaked information is bounded by some leakage parameter ℓ. The majority of the results are in the *Relative Leakage Model*, which allows the systems parameters to depend on ℓ with aim of making ℓ as large as possible relative to the length of the secret key. Akavia et al. [7] started the study of side-channel attacks in the *public-key* setting by showing Regev's encryption scheme [63] is leakage resilient in the relative leakage model. Naor and Segev [60] constructed new public-key schemes based on non-lattice assumptions, which allowed for more leakage and achieved CCA security. Katz and Vaikuntanathan [52] subsequently developed signature schemes in the relative leakage model.

The *Bounded Retrieval Model (BRM)*, introduced in [28,34], assumes a bound ℓ on the overall amount of information learned by the adversary during the entire lifetime of the system (usually by setting ℓ very large). This model differs from the relative leakage model since it ensures that all the system parameters, except the length of the secret key, are independent of ℓ. Importantly, this ensures that the *necessary* inefficiency in storage is the *only* inefficiency in the system. In particular, only a small portion of the secret key is read by honest users thus ensuring the *locality* property with little overhead (computational and/or communication) compared to a conventional system. Thus, the BRM is a natural model for constructing locally decodable and updatable non-malleable codes.

Dziembowski [34], constructed a symmetric key authenticated key agreement protocol in Random Oracle model for the BRM setting, which was subsequently extended to standard model [15]. Password authentication and secret sharing in the BRM, was studied in [28], and [37] respectively. *Non-interactive* symmetric key encryption schemes using partially compromised keys were constructed by [62] implicitly and by [29] explicitly. The first public key cryptosystems in the BRM were provided by [9] who built leakage-resilient identification schemes, leakage-resilient signature schemes (in the random oracle model), and provided tools for converting schemes in relative leakage model to the BRM. The first PKE scheme in the BRM was provided by [8] based on assumptions like lattices, quadratic residuosity and bilinear maps. Alwen et al. [10] provide an excellent survey of various leakage resilient primitives in BRM.

Other Models There are several other models of leakage tailored for various applications, for example, [14,32] study a restricted class where adversary learns a *subset of the bits* of the secret key. Micali and Reyzin in [58], assume that "only computation leaks information", whereas Ishai et al. [48] assume leakage and tampering on a subset of wires during evaluation of a circuit, in [29] Dodis et al. consider a model where the adversary can learn some function $f(\mathsf{sk})$ of the secret key, as long as f is computationally hard to invert.

2 Preliminaries on Random Access Machines

We consider RAM programs to be interactive stateful systems $\langle \Pi, \mathsf{state}, D\rangle$, where Π denotes a next instruction function, state denotes the current state stored in registers, and D denotes the content of memory. Upon input state and a value d, the next instruction function outputs the next instruction I and an updated state state'. The initial state of the RAM machine, state, is set to $(\mathsf{start}, *)$. We denote by $\mathsf{A}^D(x)$, the execution of RAM algorithm A with random access to array D and explicit input x. We define operator *Access* which outputs the access patterns of $\mathsf{A}^D(x)$, and denote by $\boldsymbol{I}$ the locations in D accessed by $\mathsf{A}^D(x)$. Thus, we write $\boldsymbol{I} \xleftarrow{Access} \mathsf{A}^D(x)$.

3 Preliminaries on LD Non-Malleable Codes

In this section we present a slightly modified definition of leakage resilient, locally decodable and updatable non-malleable codes (originally introduced by [27]).

Definition 1 (Locally Decodable and Updatable Code). *Let $\Sigma, \hat{\Sigma}$ be sets of strings, and $n, \hat{n}, p, q$ be some parameters. An $(n, \hat{n}, p, q)$ locally decodable and updatable coding scheme consists of three algorithms* $(\mathsf{ENC}, \mathsf{DEC}, \mathsf{UPDATE})$ *with the following syntax:*

- *The encoding algorithm* ENC *(*perhaps randomized*) takes input an n-block (in Σ) database and outputs an $\hat{n}$-block (in $\hat{\Sigma}$) codeword.*
- *The (local) decoding algorithm* DEC *takes input an index in $[n]$, reads at most p blocks of the codeword, and outputs a block of the database in Σ. The overall decoding algorithm simply outputs* $(\mathsf{DEC}(1), \mathsf{DEC}(2), \ldots, \mathsf{DEC}(n))$.
- *The (local) updating algorithm* UPDATE *(*perhaps randomized*) takes inputs an index in $[n]$ and a string in $\Sigma \cup \{\epsilon\}$, and reads/writes at most q blocks of the codeword. Here the string ϵ denotes the procedure of refreshing without changing anything.*

Let $C \in \hat{\Sigma}^{\hat{n}}$ be a codeword. For convenience, we denote $\mathsf{DEC}^C, \mathsf{UPDATE}^C$ as the processes of reading/writing individual block of the codeword, i.e. the codeword oracle returns or modifies an individual block upon a query. Recall that C is a random access memory where the algorithms can read/write to the memory C at individual different locations.

Definition 2 (Correctness). *An $(n, \hat{n}, p, q)$ locally decodable and updatable coding scheme (with respect to $\Sigma, \hat{\Sigma}$) satisfies the following properties. For any database $D = (D_1, D_2, \ldots, D_n) \in \Sigma^n$, let $C = (C_1, C_2, \ldots, C_{\hat{n}}) \leftarrow \mathsf{ENC}(D)$ be a codeword output by the encoding algorithm. Then we have:*

- *for any index $i \in [n]$, $\Pr[\mathsf{DEC}^C(i) = D_i] = 1$, where the probability is over the randomness of the encoding algorithm.*
- *for any update procedure with input $(j, D') \in [n] \times \Sigma \cup \{\epsilon\}$, let C' be the resulting codeword by running $\mathsf{UPDATE}^C(j, D')$. Then we have $\Pr[\mathsf{DEC}^{C'}(j) = D'] = 1$, where the probability is over the encoding and update procedures. Moreover, the decodings of the other positions remain unchanged.*

Following [27], our definition includes a third party called the *updater*, who reads the underlying messages and decides how to update the codeword. This notion captures the RAM program computing on the underlying, unencoded data. The model allows the adversary to learn the location that the updater updated the messages, but not the content of the updated messages. The security guarantee is that an adversary who launches tampering and leakage attacks when the updater is interacting with the codeword cannot learn anything about the underlying encoded messages (or the updated messages during the interaction).

Our experiment is interactive and consists of rounds: The adversary adaptively chooses two rounds i, j such that $i \leq j$, submits a leakage function in round i, gets its output and then submits a tampering function in round j. We assume WLOG that at the end of each round, the updater runs UPDATE, and the codeword will be somewhat updated and refreshed. The security experiment then considers the decoding of the entire message after each round.

Definition 3 (One Time Tampering and Leakage Experiment). *Let λ be the security parameter, $\mathcal{F}, \mathcal{G}$ be some families of functions. Let $(\mathsf{ENC}, \mathsf{DEC}, \mathsf{UPDATE})$ be an $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$. Let $\mathcal{U}$ be an updater that takes input a database $D \in \Sigma^n$ and outputs an index $i \in [n]$ and $v \in \Sigma$. Flags Leaked and $\mathsf{Tampered}$ will be set to 0 and let r be the total number of rounds. Then for any blocks of databases $D = (D_1, D_2, \ldots, D_n) \in \Sigma^n$, and any (non-uniform) adversary $\mathcal{A}$, any updater $\mathcal{U}$ define the following experiment $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D}$:*

- *The challenger first computes an initial encoding $C^{(0)} \leftarrow \mathsf{ENC}(D)$.*
- *Then the following procedure repeats, at each round j, let $C^{(j)}$ be the current codeword and $D^{(j)}$ be the underlying database:*
  - *The updater computes $(i^{(j)}, v) \leftarrow \mathcal{U}(D^{(j)})$ for the challenger. The challenger runs $\mathsf{UPDATE}^{C^{(j)}}(i^{(j)}, v)$. Let $D^{(j+1)}$ denote the updated database.*
  - *$\mathcal{A}$ sends either a tampering function $f \in \mathcal{F}$ and/or a leakage function $g \in \mathcal{G}$ or $\perp$ to the challenger.*
  - *if Leaked is 0 and g is sent by $\mathcal{A}$, the challenger sends back a leakage $\ell = g(C^{(j+1)})$ and sets Leaked to 1.*
  - *if Leaked is 1, $\mathsf{Tampered}$ is 0 and f is sent by $\mathcal{A}$, the challenger replaces the codeword with $f(C^{(j+1)})$ and sets $\mathsf{Tampered}$ to 1.*
  - *We define $D^{(j+1)} \stackrel{\text{def}}{=} \left(\mathsf{DEC}^{f(C^{(j+1)})}(1), \ldots, \mathsf{DEC}^{f(C^{(j+1)})}(n)\right)$.*
  - *$\mathcal{A}$ may terminate the procedure at any point.*
- *Let r be the total number of rounds above. At the end, the experiment outputs*

$$\left(\ell, D^{(0)}, \ldots, D^{(r)}\right).$$

Definition 4 (One Time Non-malleability and Leakage Resilience against Attacks). *An $(n, \hat{n}, p, q)$-locally decodable and updatable coding scheme with respect to $\Sigma, \hat{\Sigma}$ is non-malleable against $\mathcal{F}$ and leakage resilient against $\mathcal{G}$ if for all* PPT *(non-uniform) adversaries $\mathcal{A}$, and* PPT *updater $\mathcal{U}$, there exists some* PPT *(non-uniform) simulator $\mathcal{S}$ such that for any $D = (D_1, \ldots, D_n) \in \Sigma^n$, $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D}$ is (computationally) indistinguishable to the following ideal experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},D}$:*

- *The experiment proceeds in rounds. Let $D^{(0)} = D$ be the initial database.*
- *At each round j, the experiment runs the following procedure:*
  - *At the beginning of each round, the updater runs $(i^{(j)}, v) \leftarrow \mathcal{U}(D^{(j)})$ and sends the index $i^{(j)}$ to the simulator. Then the experiment updates $D^{(j+1)}$ as follows: set $D^{(j+1)} := D^{(j)}$ for all coordinates except $i^{(j)}$, and set $D^{(j+1)}[i^{(j)}] := v$.*
  - *$\mathcal{S}$ outputs $(\mathcal{I}^{(j+1)}, \boldsymbol{w}^{(j+1)})$, where $\mathcal{I}^{(j+1)} \subseteq [n]$.*
  - *Define*

$$D^{(j+1)} = \begin{cases} \boldsymbol{w}^{(j+1)} & \textit{if } \mathcal{I}^{(j+1)} = [n] \\ D^{(j+1)}|_{\mathcal{I}^{(j+1)}} := \perp, D^{(j+1)}|_{\bar{\mathcal{I}}^{(j+1)}} := D^{(j+1)}|_{\bar{\mathcal{I}}^{(j+1)}} & \textit{otherwise,} \end{cases}$$

where $\boldsymbol{x}|_{\mathcal{I}}$ denotes the coordinates $\boldsymbol{x}[v]$ where $v \in \mathcal{I}$, and the bar denotes the complement of a set.

- *Let r be the total number of rounds above. At the end, $\mathcal{S}$ outputs ℓ and the experiment outputs*

$$\left(\ell, D^{(0)}, \ldots, D^{(r)}\right).$$

4 Preliminaries on Primitives in the BRM

A public key encryption scheme ($\mathcal{E}$) in the BRM consists of the algorithms (KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$), which are all parameterized by a security parameter λ and a leakage parameter ℓ. The syntax and correctness property of an encryption scheme follow the standard notion of public-key encryption. We define the following CPA game, with leakage ℓ, between an adversary $\mathcal{A}$ and a challenger.

- **Key Generation:** The challenger computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\ell)$ and gives pk to the adversary $\mathcal{A}$.
- **Leakage:** The adversary $\mathcal{A}$ selects a PPT function $g : \{0,1\}^* \to \{0,1\}^\ell$ and gets $g(\mathsf{sk})$ from the challenger.
- **Challenge:** The adversary $\mathcal{A}$ selects two messages m_0, m_1. The challenger chooses $b \leftarrow \{0,1\}$ uniformly at random and gives $c \leftarrow \mathsf{Encrypt}(m_b, \mathsf{pk})$ to the adversary $\mathcal{A}$.
- **Output:** The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. We say that $\mathcal{A}$ wins the game if $b' = b$.

For any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is defined as $\mathbf{Adv}^{\mathrm{CPA}}_{\mathcal{E},\mathcal{A}}(\lambda, \ell) \overset{\mathrm{def}}{=} |\Pr[\mathcal{A} \text{ wins}] - \frac{1}{2}|$.

Definition 5 (Leakage-Resilient PKE). *[8] A public-key encryption scheme $\mathcal{E}$ is leakage-resilient, if for any polynomial $\ell(\lambda)$ and any PPT adversary $\mathcal{A}$, we have $\mathbf{Adv}^{\mathrm{CPA}}_{\mathcal{E},\mathcal{A}}(\lambda, \ell(\lambda)) = \mathsf{negl}(\lambda)$.*

Definition 6 (PKE in the BRM). *[8] We say that a leakage-resilient PKE scheme is a **PKE in the BRM**, if the public-key size, ciphertext size, encryption-time and decryption-time (and the number of secret-key bits read by decryption) are independent of the leakage-bound ℓ. More formally, **there exist** polynomials $\mathsf{pksize}, \mathsf{ctsize}, \mathsf{encT}, \mathsf{decT}$, such that, **for any** polynomial ℓ and any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\ell)$, $m \in \mathcal{M}$, $c \leftarrow \mathsf{Encrypt}(m, \mathsf{pk})$, the scheme satisfies:*

1. *Public-key size is $|\mathsf{pk}| \leq O(\mathsf{pksize}(\lambda))$, ciphertext size is $|c| \leq O(\mathsf{ctsize}(\lambda, |m|))$.*
2. *Run-time of $\mathsf{Encrypt}(m, \mathsf{pk})$ is $\leq O(\mathsf{encT}(\lambda, |m|))$.*
3. *Run-time of $\mathsf{Decrypt}(c, \mathsf{sk})$, and the number of bits of sk accessed is, $\leq O(\mathsf{decT}(\lambda, |m|))$.*

*The **relative-leakage** of the scheme is $\alpha \overset{def}{=} \frac{\ell}{|\mathsf{sk}|}$.*

Alwen et al. [8] give a generic transformation to construct a CCA-secure PKE scheme in the BRM using Naor-Young "double encryption" paradigm.

A signature scheme consists of three algorithms: ($\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify}$). Attacker is separated into two parts, first part, $\mathcal{A}_1$, will interact with leakage oracle and signing oracle and output an arbitrary hint for the second part, $\mathcal{A}_2$. $\mathcal{A}_2$ only accesses signature oracle and tries to forge the signature of a message. The Entropic Unforgeability attack Game EUG^λ_ℓ is as follows:

1. Challenger select $(\mathsf{vk}, help, \mathsf{sk}_\sigma) \leftarrow \mathsf{Gen}(1^\lambda)$ and gives vk to $\mathcal{A}_1$.
2. Adversary $\mathcal{A}_1$ is given access to signing oracle $\mathcal{S}_{\mathsf{sk}_\sigma}(\cdot)$ and leakage oracle $\mathcal{O}^{\lambda,\ell}_{\mathsf{sk}_\sigma}(.)$ and outputs a hint $v \in \{0,1\}^*$.
3. Adversary $\mathcal{A}_2$ is given access to hint v and signing oracle $\mathcal{S}_{\mathsf{sk}_\sigma}(\cdot)$ and outputs message, signature pair (m, σ).

We define the advantage of the attacker $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to be the probability that $\mathsf{Verify}_{\mathsf{vk}}(m, \sigma) = 1$ and that the signing oracle was never queried with m.

Definition 7 (Signature Scheme). *[9] Let $View_{\mathcal{A}_1}$ be a random variable denoting the view of $\mathcal{A}_1$ which includes its random coin and the responses it gets from signing oracle and leakage oracle. Let $MSG_{\mathcal{A}_2}$ be a random variable of the messages output by $\mathcal{A}_2$ in EUG^λ_ℓ. Adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is entropic if $\tilde{H}_\infty(MSG_{\mathcal{A}_2} \mid View_{\mathcal{A}_1}) \geq \lambda$ for security parameter λ. We say that a signature scheme is entropically unforgeable with leakage ℓ if the advantage of adversary $\mathcal{A}$ in EUG^λ_ℓ is negligible in λ.*

Remark 1. Entropic signatures in the BRM can be constructed in the random oracle (RO) model (cf. [9,10]). Combining a signature scheme in the relative leakage model (with additional properties) such as [52] with the leakage amplification techniques of [9], it may be possible to construct entropic signatures in the BRM without RO. However, as discussed in Remark 2 in Section 7.1, the primitive we require is slightly weaker than regular entropic signatures in the BRM and can be constructed in a straightforward manner in the standard model, without RO. Nevertheless, for conceptual simplicity we present our constructions and state our theorems in terms of the existence of entropic signatures in the BRM.

5 Preliminaries on IB-HPS

Definition 8 (IB-HPS). *[8] An* Identity-Based Hash Proof System *(*IB-HPS*) consists of PPT algorithms: (*$\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Encap}^*, \mathsf{Decap}$*). The algorithms have the following syntax.*

- $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$*: The setup algorithm takes as input a security parameter λ and produces the* master public key mpk *and the* master secret key msk. *The master public key defines an* identity set $\mathcal{ID}$*, and an* encapsulated-key set $\mathcal{K}$.
  All other algorithms $\mathsf{KeyGen}, \mathsf{Encap}, \mathsf{Decap}, \mathsf{Encap}^*$ *implicitly include* mpk *as an input.*
- $\mathsf{sk}_{\mathsf{ID}} \leftarrow \mathsf{KeyGen}(\mathsf{ID}, \mathsf{msk})$*: For any identity* $\mathsf{ID} \in \mathcal{ID}$*, the* KeyGen *algorithm used the master secret key* msk *to sample an identity secret key* $\mathsf{sk}_{\mathsf{ID}}$.
- $(c, k) \leftarrow \mathsf{Encap}(\mathsf{ID})$*: The* valid *encapsulation algorithm creates pairs (c, k) where c is a valid ciphertext, and $k \in \mathcal{K}$ is the encapsulated-key.*
- $c \leftarrow \mathsf{Encap}^*(\mathsf{ID})$*: The alternative* invalid *encapsulation algorithm which samples an invalid ciphertext c.*
- $k \leftarrow \mathsf{Decap}(c, \mathsf{sk}_{\mathsf{ID}})$*: The decapsulation algorithm is deterministic, and takes an identity secret key* $\mathsf{sk}_{\mathsf{ID}}$ *and a ciphertext c and outputs the encapsulated key k.*

Following [8], we require the IB-HPS which satisfy the following properties.

1. *Correctness of Decapsulation:* For any values of $\mathsf{mpk}, \mathsf{msk}$ produced by $\mathsf{Setup}(1^\lambda)$, any $\mathsf{ID} \in \mathcal{ID}$ we have:
$$\Pr\left[k \neq k' | \mathsf{sk}_{\mathsf{ID}} \leftarrow \mathsf{KeyGen}(\mathsf{ID}, \mathsf{msk}), (c, k) \leftarrow \mathsf{Encap}(\mathsf{ID}),\ k' \leftarrow \mathsf{Decap}(c, \mathsf{sk}_{\mathsf{ID}})\right] \leq \mathsf{negl}(\lambda)$$
2. *Valid/Invalid Ciphertext Indistinguishability:* The valid ciphertexts generated by Encap and the invalid ciphertexts generated by Encap^* should be indistinguishable *even given the identity secret key.* In particular, we define the following distinguishability game between an adversary $\mathcal{A}$ and a challenger.
$$\mathsf{VI\text{-}IND}(\lambda)$$
   - **Setup:** The challenger computes $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ and gives mpk to the adversary $\mathcal{A}$.
   - **Test Stage 1:** The adversary $\mathcal{A}$ adaptively queries the challenger with $\mathsf{ID} \in \mathcal{ID}$ and the challenger responds with $\mathsf{sk}_{\mathsf{ID}}$.
   - **Challenge Stage:** The adversary $\mathcal{A}$ selects an *arbitrary* challenge identity $\mathsf{ID}^* \in \mathcal{ID}$.
     The challenger chooses $b \leftarrow \{0, 1\}$.
     If $b = 0$ the challenger computes $(c, k) \leftarrow \mathsf{Encap}(\mathsf{ID}^*)$.
     If $b = 1$ the challenger computes $c \leftarrow \mathsf{Encap}^*(\mathsf{ID}^*)$.
     The challenger gives c to the adversary $\mathcal{A}$.
   - **Test Stage 2:** The adversary $\mathcal{A}$ adaptively queries the challenger with $\mathsf{ID} \in \mathcal{ID}$ and the challenger responds with $\mathsf{sk}_{\mathsf{ID}}$.
   - **Output:** The adversary $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ which is the output of the game. We say that $\mathcal{A}$ *wins* the game if $b' = b$.

   Note: In test stages 1,2 the challenger computes $\mathsf{sk}_{\mathsf{ID}} \leftarrow \mathsf{KeyGen}(\mathsf{ID}, \mathsf{msk})$ the first time that ID is queried and responds to all future queries on the same ID with the same $\mathsf{sk}_{\mathsf{ID}}$.
   Note that, during the challenge phase, the adversary can choose *any* identity ID^*, and possibly even one for which it has seen the secret key $\mathsf{sk}_{\mathsf{ID}^*}$ in Test Stage 1 (or the adversary can simply get $\mathsf{sk}_{\mathsf{ID}^*}$ in Test Stage 2). We define the advantage of $\mathcal{A}$ in distinguishing valid/invalid ciphertexts to be $\mathbf{Adv}^{\text{VI-IND}}_{\mathsf{IB\text{-}HPS},\mathcal{A}}(\lambda) \overset{\text{def}}{=} |\Pr[\mathcal{A}\,\text{wins}] - \frac{1}{2}|$. We require that $\mathbf{Adv}^{\text{VI-IND}}_{\mathsf{IB\text{-}HPS},\mathcal{A}}(\lambda) = \mathsf{negl}(\lambda)$.

3. *Universality/Smoothness/Leakage-Smoothness:* Other than properties 1 and 2, we will need one additional information theoretic property. Essentially, we want to ensure that there are many possibilities for the decapsulation of an *invalid* ciphertext, which are left undetermined by the public parameters of the system. We define three flavors of this property as follows.

   Definition 9 (Universal $\mathsf{IB\text{-}HPS}$). *[8] We say that an* $\mathsf{IB\text{-}HPS}$ *is* (m, ρ)***-universal*** *if, for any fixed values of* $\mathsf{mpk}, \mathsf{msk}$ *produced by* $\mathsf{Setup}(1^\lambda)$*, and any fixed* $\mathsf{ID} \in \mathcal{ID}$ *the following two properties hold:*
   (a) Let SK *be a random variable for the output of* $\mathsf{KeyGen}(\mathsf{ID}, \mathsf{msk})$*. Then* $\boldsymbol{H}_\infty(\mathsf{SK}) \geq m$*.*
   (b) For any fixed distinct values $\mathsf{sk_{ID}} \neq \mathsf{sk'_{ID}}$ *in the support of* SK*, we have*

$$\Pr_{c \leftarrow \mathsf{Encap}^*(\mathsf{ID})}[\mathsf{Decap}(c, \mathsf{sk_{ID}}) = \mathsf{Decap}(c, \mathsf{sk'_{ID}})] \leq \rho$$

   Notice the significant difference between valid and invalid ciphertexts. For valid ciphertexts c, the correctness of decapsulation ensures that there is a single value $k \in \mathcal{K}$ such that $\mathsf{Decap}(c, \mathsf{sk_{ID}}) = k$ for (virtually) all choices of $\mathsf{sk_{ID}}$ (of which there are many by (a)). On the other hand, for invalid ciphetexts c, (b) ensures that it is highly unlikely that any two distinct secret-keys $\mathsf{sk_{ID}}$ will decapsulate c to the same value k.

   Definition 10 (Smooth/Leakage-Smooth $\mathsf{IB\text{-}HPS}$). *[8] We say that an* $\mathsf{IB\text{-}HPS}$ *is* ***smooth*** *if, for any fixed values of* $\mathsf{mpk}, \mathsf{msk}$ *produced by* $\mathsf{Setup}(1^\lambda)$*, and any* $\mathsf{ID} \in \mathcal{ID}$*, we have:*

$$\boldsymbol{SD}((c, k), (c, k')) \leq \mathsf{negl}(\lambda)$$

   where $c \leftarrow \mathsf{Encap}^*(\mathsf{ID})$*,* $k' \leftarrow U_\mathcal{K}$ *and* k *is sampled by choosing* $\mathsf{sk_{ID}} \leftarrow \mathsf{KeyGen}(\mathsf{ID}, \mathsf{msk})$ *and computing* $k = \mathsf{Decap}(c, \mathsf{sk_{ID}})$*. We say that an* $\mathsf{IB\text{-}HPS}$ *is* ℓ***-leakage-smooth*** *if, for any (possibly randomized) function* $f(\cdot)$ *with* ℓ*-bit output, we have:*

$$\boldsymbol{SD}((c, f(\mathsf{sk_{ID}}), k), (c, f(\mathsf{sk_{ID}}), k')) \leq \mathsf{negl}(\lambda)$$

   where $c, k, \mathsf{sk_{ID}}, k'$ *are sampled as above. Note, for this property* f *need not be efficient.*

   We note that, [8] present construction of leakage smooth $\mathsf{IB\text{-}HPS}$.

6 Split-State Public Key Encryption in the BRM ($\mathsf{SS\text{-}BRM\text{-}PKE}$)

In this section, we define a new primitive called as Split-State Public Key Encryption in Bounded Retrieval Model ($\mathsf{SS\text{-}BRM\text{-}PKE}$). Intuitively, $\mathsf{SS\text{-}BRM\text{-}PKE}$ is a public key encryption scheme with following properties:

1. The secret key sk is stored in the split-state $\mathsf{sk}_1||\mathsf{sk}_2$ and the adversary is allowed to obtain leakage on sk_1 and sk_2 independently.
2. The encryption scheme is secure in the bounded retrieval model, as defined in Definition 6 with respect to the split-state leakage.
3. The encryption scheme is chosen plaintext attack (CPA)-secure even in the presence of adversary who can observe the access pattern of the blocks of pk and $\mathsf{sk}_1||\mathsf{sk}_2$ accessed during encryption and decryption procedure.
4. The scheme is CPA-secure against an adversary getting additional split-state leakage (bounded) on the secret key, even *after* receiving the ciphertext.

Formally, a public key encryption scheme ($\mathcal{E}$) in the SS-BRM consists of the algorithms (KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$), which are all parameterized by a security parameter λ and a leakage parameter ℓ. The syntax and correctness property of an encryption scheme follow the standard notion of public-key encryption. We define the following semantic-security game ($\mathsf{SS\text{-}BRM\text{-}PKE\text{-}CPA}$) with leakage ℓ between an adversary $\mathcal{A}$ and a challenger.

- **Key Generation:** The challenger computes $(\mathsf{pk}, \mathsf{sk}_1||\mathsf{sk}_2) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\ell)$ and gives pk to the adversary $\mathcal{A}$.
- **Message Commitment:** The adversary $\mathcal{A}$ selects two messages m_0, m_1.
- **Pre-challenge Leakage:** The adversary $\mathcal{A}$ selects a PPT function $g := (g_1, g_2) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^\ell \times \{0,1\}^\ell$ and gets $L_1 := g_1(\mathsf{sk}_1)$, $L_2 := g_2(\mathsf{sk}_2)$ from the challenger.
- **Challenge:** The challenger chooses $b \leftarrow \{0,1\}$ uniformly at random and gives $c \leftarrow \mathsf{Encrypt}(m_b, \mathsf{pk})$ to the adversary $\mathcal{A}$.
- **Encryption Access Patterns:** The challenger also sends the access pattern $(i^{(1)}, i^{(2)}, \ldots, i^{(t)})$ corresponding to the encryption procedure, to $\mathcal{A}$.
- **Post-challenge Leakage:** The adversary $\mathcal{A}$ selects a PPT function $g' := (g'_1, g'_2) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^\ell \times \{0,1\}^\ell$ and gets $L'_1 := g'_1(L_1, L_2, c, \mathsf{sk}_1)$, $L'_2 := g'_2(L_1, L_2, c, \mathsf{sk}_2)$ from the challenger.
- **Decryption Access Patterns:** The challenger also sends the access pattern $(S^1, S^2) \xleftarrow{Access} \mathsf{Decrypt}^{\mathsf{sk}_1||\mathsf{sk}_2}(c)$, to $\mathcal{A}$. S^i is a set of indices s^i_j of sk_i for $i \in \{1,2\}$ and $j \in [n]$, where $|\mathsf{sk}_1| = |\mathsf{sk}_2| = n$. Also, $|S^i| = t$, where t is the number of locations of sk_1 and sk_2 required to be accessed to decrypt any ciphertext.
- **Output:** The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. We say that $\mathcal{A}$ wins the game if $b' = b$.

For any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is defined as $\mathbf{Adv}^{\mathsf{SS\text{-}BRM\text{-}PKE\text{-}CPA}}_{\mathcal{E},\mathcal{A}}(\lambda, \ell) \overset{\text{def}}{=} |\Pr[\mathcal{A}\,\text{wins}] - \frac{1}{2}|$. It should be noted that the decryption access patterns indicating which parts of secret key were accessed during the decryption must be provided to the adversary only *after* the leakage information, otherwise the adversary can simply ask for the leakage on the secret key positions "relevant" to the decryption of the challenge ciphertext.

The very high-level idea is to do a double encryption $\mathsf{Encrypt}_{\mathsf{pk}_1}(\mathsf{Encrypt}_{\mathsf{pk}_2}(m))$, where $\mathsf{Encrypt}$ is instantiated using the encryption scheme based on leakage-smooth IB-HPS given by [8] and $\mathsf{pk}_1, \mathsf{pk}_2$ are two independent public keys with corresponding secret keys $\mathsf{sk}_1, \mathsf{sk}_2$. Intuitively, this allows us to deal with post-challenge leakage, because, using the properties of the first leakage-smooth IB-HPS, we move to a hybrid where, given the pre-challenge leakage and the challenge ciphertext $c_1 := \mathsf{Encrypt}_{\mathsf{pk}_1}(\mathsf{Encrypt}_{\mathsf{pk}_2}(m_b))$, where $b \in \{0,1\}$, we have that $c_2 := \mathsf{Encrypt}_{\mathsf{pk}_2}(m)$ is information-theoretically hidden. This means that the post-challenge leakage can be converted to pre-challenge leakage on sk_2 and by using the security properties of the second leakage-smooth IB-HPS, we cannot distinguish encryptions of m_0 from m_1.

6.1 Construction of SS-BRM-PKE

Let, $\mathcal{E} := (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be the encryption scheme and let
$\mathcal{P}_1 := (\mathsf{Setup}_1, \mathsf{KeyGen}_1, \mathsf{Encap}_1, \mathsf{Encap}^*_1, \mathsf{Decap}_1)$, $\mathcal{P}_2 := (\mathsf{Setup}_2, \mathsf{KeyGen}_2, \mathsf{Encap}_2, \mathsf{Encap}^*_2, \mathsf{Decap}_2)$
be the ℓ-leakage-smooth IB-HPS with leakage-amplification as proposed by [8].

- $\mathsf{KeyGen} := (\mathsf{Setup}_1, \mathsf{Setup}_2, \mathsf{KeyGen}_1, \mathsf{KeyGen}_2)$: Generates $(\mathsf{mpk}_1, \mathsf{msk}_1) \leftarrow \mathsf{Setup}_1(1^\lambda)$ and Generates $(\mathsf{mpk}_2, \mathsf{msk}_2) \leftarrow \mathsf{Setup}_2(1^\lambda)$. The master public keys mpk_1 and mpk_2, define the public key sets $\mathcal{PK}_1$ and $\mathcal{PK}_2$ respectively. All other algorithms KeyGen_i, Encap_i, Encap^*_i, Decap_i implicitly include mpk_i as an input, for $i \in \{1,2\}$.
  $\mathsf{sk}_1 \leftarrow \mathsf{KeyGen}_1(\mathsf{pk}_1, \mathsf{msk}_1)$: For any $\mathsf{pk}_1 \in \mathcal{PK}_1$. $\mathsf{sk}_2 \leftarrow \mathsf{KeyGen}_2(\mathsf{pk}_2, \mathsf{msk}_2)$: For any $\mathsf{pk}_2 \in \mathcal{PK}_2$.
  Sets $\mathsf{pk} := \mathsf{pk}_1||\mathsf{pk}_2$ and output $(\mathsf{pk}, sk_1||\mathsf{sk}_2)$.
- $\mathsf{Encrypt}^{\mathsf{pk}}(m)$: Computes $(c^1_2, k_2) \leftarrow \mathsf{Encap}_2(\mathsf{pk}_2)$ and sets $c^2_2 := m \oplus k_2$.
  Computes $(c^1_1, k_1) \leftarrow \mathsf{Encap}_1(\mathsf{pk}_1)$ and sets $c^2_1 := (c^1_2||c^2_2) \oplus k_1$.
  Set $c := (c^1_1, c^2_1)$, and output c along with the access patterns of $\mathsf{Encrypt}^{\mathsf{pk}}(m)$.
- $\mathsf{Decrypt}^{\mathsf{sk}_1||\mathsf{sk}_2}(c)$: Computes $k_1 \leftarrow \mathsf{Decap}_1(c^1_1, \mathsf{sk}_1)$. Computes $(c^1_2||c^2_2) := c^2_1 \oplus k_1$.
  Computes $k_2 \leftarrow \mathsf{Decap}_2(c^1_2, \mathsf{sk}_2)$. Computes $m := c^2_2 \oplus k_2$.
  Output m.

Note that to achieve locality that is polylogarithmic in λ, we require a sub-exponentially-hard PRG, prg, and replace $c^2_1 := (c^1_2||c^2_2) \oplus k_1$ with $c^2_1 := (c^1_2||c^2_2) \oplus \mathsf{prg}(k_1)$ and $c^2_2 := m \oplus k_2$ with $c^2_2 := m \oplus \mathsf{prg}(k_2)$. For simplicity we keep the presentation as above.

Theorem 2. *If $\mathcal{P}_1$ and $\mathcal{P}_2$ are l-leakage smooth IB-HPS then $\mathcal{E}$ is a l-leakage resilient SS-BRM-PKE.*

We now show that the $\mathbf{Adv}^{\mathsf{SS\text{-}BRM\text{-}PKE}}_{\mathcal{E},\mathcal{A}}(\lambda, \ell)$ is $\mathsf{negl}(\lambda)$ via a series of hybrids. Note that, the size of pk is independent of leakage and is usually short. Therefore, we can consider that entire public key pk is read during encryption procedure and hence omit outputting encryption access pattern to the adversary without loss of generality. In other words, the encryption access pattern consists of reading entire pk for every encryption.

Proof (of Theorem 2).

Hybrid H_0: This is the detailed step-wise experiment:

- **Key Generation:** Generate the keys $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{Gen}(1^\lambda)$ and $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{Gen}(1^\lambda)$. Output the public keys pk_1 and pk_2.
- **Message Commitment:** Receive the messages m_0 and m_1 from the adversary.
- **Pre-challenge Leakage:** Compute the pre-challenge leakage $L_1 := g_1(\mathsf{sk}_1)$, $L_2 := g_2(\mathsf{sk}_2)$. Output L_1 and L_2 to the adversary.
- **Challenge:**
  1. Compute $(c_1^1, k_1) \leftarrow \mathsf{Encap}_1(\mathsf{pk}_1)$ and $(c_2^1, k_2) \leftarrow \mathsf{Encap}_2(\mathsf{pk}_2)$.
  2. Compute $c_2^2 := m_b \oplus k_2$ and $c_1^2 := (c_2^1 || c_2^2) \oplus k_1$. Set $c_1 := (c_1^1, c_1^2)$ and $c_2 := (c_2^1, c_2^2)$.
  3. Output c_1 as challenge ciphertext.
- **Post-challenge Leakage:** Compute the post-challenge leakage $L_1' := g_1'(L_1, L_2, c_1, \mathsf{sk}_1)$, $L_2' := g_2'(L_1, L_2, c_1, \mathsf{sk}_2)$. Output L_1' and L_2'.
- **Decryption Access Patterns:** Compute the decryption access patterns $(S^1, S^2) \xleftarrow{Access} \mathsf{Decrypt}^{\mathsf{sk}_1||\mathsf{sk}_2}(c_1)$. Output (S^1, S^2).
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (S^1, S^2))$

Claim 6.1. The adversarial view in hybrid H_0 is identical to its view in the experiment $\mathsf{SS\text{-}BRM\text{-}PKE}$.

Proof. This claim can be verified by simple observation.

Hybrid H_1: This experiment differs from H_0 in only one step where instead of using the keys k_1 and k_2 derived from encapsulation we use the keys $\widetilde{k_1}$ and $\widetilde{k_2}$; derived from decapsulation; for encryption.

- **Challenge:**
  1. Compute $(c_1^1, k_1) \leftarrow \mathsf{Encap}_1(\mathsf{pk}_1)$ and $(c_2^1, k_2) \leftarrow \mathsf{Encap}_2(\mathsf{pk}_2)$.
  2. $\underline{\text{Compute } \widetilde{k_1} = \mathsf{Decap}(\mathsf{sk}_1, c_1^1) \text{ and } \widetilde{k_2} = \mathsf{Decap}(\mathsf{sk}_2, c_2^1).}$
  3. Compute $c_2^2 := m_b \oplus \widetilde{k_2}$ and $c_1^2 := (c_2^1 || c_2^2) \oplus \widetilde{k_1}$. Set $c_1 := (c_1^1, c_1^2)$ and $c_2 := (c_2^1, c_2^2)$.
  4. Output c_1 as challenge ciphertext.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (S^1, S^2))$

Claim 6.2. The adversarial view in hybrid H_1 is statistically indistinguishable from its view in hybrid H_0.

Proof. This claim is true because of the correctness property of the IB-HPS $\mathcal{P}_1$ and $\mathcal{P}_2$.

Hybrid H_2: This experiment differs from H_1 in only one step where instead of using valid encapsulation, we use invalid encapsulation to generate $\widetilde{c}_1^1$ and $\widetilde{c}_2^1$.

- **Challenge:**
  1. $\underline{\text{Compute } (\widetilde{c}_1^1) \leftarrow \mathsf{Encap}_1^*(\mathsf{pk}_1) \text{ and } (\widetilde{c}_2^1) \leftarrow \mathsf{Encap}_2^*(\mathsf{pk}_2).}$
  2. Compute $\widetilde{k_1} = \mathsf{Decap}(\mathsf{sk}_1, \widetilde{c}_1^1)$ and $\widetilde{k_2} = \mathsf{Decap}(\mathsf{sk}_2, \widetilde{c}_2^1)$.
  3. Compute $c_2^2 := m_b \oplus \widetilde{k_2}$ and $c_1^2 := (\widetilde{c}_2^1 || c_2^2) \oplus \widetilde{k_1}$. Set $c_1 := (\widetilde{c}_1^1, c_1^2)$ and $c_2 := (\widetilde{c}_2^1, c_2^2)$.
  4. Output c_1 as challenge ciphertext.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (S^1, S^2))$

Claim 6.3. The adversarial view in hybrid H_2 is computationally indistinguishable from its view in hybrid H_1.

Proof. This claim is true because of the valid/invalid indistinguishability property of the IB-HPS $\mathcal{P}_1$ and $\mathcal{P}_2$. Note that indistinguishability holds *even given the secret keys* $\mathsf{sk}_1, \mathsf{sk}_2$.

Hybrid H_3: This experiment differs from H_2 in that the secret key sk_1 gets re-sampled.

- **Post-challenge Leakage:** Let, for a fixed public key pk_1, the random variable $\mathcal{SK}_1$ denote the distribution over the secret keys corresponding to pk_1 from which the key generation algorithm Gen samples the secret key sk_1. Also, consider the following function $f(\mathcal{SK}_1) := (g_1(\mathcal{SK}_1), \mathsf{Decap}(c_1, \mathcal{SK}_1))$. Sample a new secret key $\widetilde{\mathsf{sk}_1}$ from $\mathcal{SK}_1 | f(\mathcal{SK}_1) = (L_1, \widetilde{k_1})$.

  Compute the post-challenge leakage $L_1' := g_1'(L_1, L_2, c_1, \widetilde{\mathsf{sk}_1})$, $L_2' := g_2'(L_1, L_2, c_1, \mathsf{sk}_2)$. Output L_1' and L_2'.
- **Decryption Access Patterns:** Compute the decryption access patterns $(\tilde{S}^1, S^2) \xleftarrow{Access} \mathsf{Decrypt}^{\widetilde{\mathsf{sk}_1}||\mathsf{sk}_2}(c_1)$. Output $(\tilde{S}^1, S^2)$.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (\tilde{S}^1, S^2))$

Claim 6.4. The adversarial view in hybrid H_3 is identical to its view in hybrid H_2.

Proof. H_3 is simply a re-writing of H_2, where secret key $\widetilde{\mathsf{sk}_1}$ is re-sampled, conditioned on all information released about sk_1 thus far.

Hybrid H_4: This experiment differs from H_3 in only one step where instead of using $\widetilde{k}_1$ we use a uniform random key $\widehat{k}_1$.

- **Challenge:**
  1. Compute $(\widetilde{c}_1^1) \leftarrow \mathsf{Encap}_1^*(\mathsf{pk}_1)$ and $(\widetilde{c}_2^1) \leftarrow \mathsf{Encap}_2^*(\mathsf{pk}_2)$.
  2. $\underline{\text{Sample } \widehat{k_1} \leftarrow U_{\mathcal{K}}}$ and compute $\widetilde{k_2} = \mathsf{Decap}(\mathsf{sk}_2, \widetilde{c}_2^1)$.
  3. Compute $c_2^2 := m_b \oplus \widetilde{k_2}$ and $c_1^2 := (\widetilde{c}_2^1 || c_2^2) \oplus \widehat{k}_1$. Set $c_1 := (\widetilde{c}_1^1, c_1^2)$ and $c_2 := (\widetilde{c}_2^1, c_2^2)$.
  4. Output c_1 as challenge ciphertext.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (\tilde{S}^1, S^2))$

Claim 6.5. The adversarial view in hybrid H_4 is statistically indistinguishable from its view in hybrid H_3.

Proof. The claim is true because of the leakage smoothness property of the IB-HPS $\mathcal{P}_1$, which allows us to replace $\widetilde{k}_1$ with a uniform random value $(\widehat{k}_1)$.

Hybrid H_5: This experiment differs from H_4 in only one step where we replace ciphertext by a uniform random value.

- **Challenge:**
  1. Compute $(\widetilde{c}_1^1) \leftarrow \mathsf{Encap}_1^*(\mathsf{pk}_1)$ and $(\widetilde{c}_2^1) \leftarrow \mathsf{Encap}_2^*(\mathsf{pk}_2)$.
  2. Compute $\widetilde{k}_2 = \mathsf{Decap}(\mathsf{sk}_2, \widetilde{c}_2^1)$.
  3. Compute $c_2^2 := m_b \oplus \widetilde{k}_2$ and $\underline{\widetilde{c}_1^2 \leftarrow U}$. Set $c_1 := (\widetilde{c}_1^1, \widetilde{c}_1^2)$ and $c_2 := (\widetilde{c}_2^1, c_2^2)$.
  4. Output c_1 as challenge ciphertext.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (\tilde{S}^1, S^2))$

Claim 6.6. The adversarial view in hybrid H_5 is identical to its view in hybrid H_4.

Proof. This is trivial observation.

Hybrid H_6: This experiment differs from H_5 in only the order of execution of challenge and post-challenge leakage steps.

- **Challenge 1:**
  1. Compute $(\widetilde{c}_1^1) \leftarrow \mathsf{Encap}_1^*(\mathsf{pk}_1)$.
  2. Sample $\widetilde{c}_1^2 \leftarrow U$. Set $c_1 := (\widetilde{c}_1^1, \widetilde{c}_1^2)$.
  3. Output c_1 as challenge ciphertext.
- **Post-challenge Leakage 1:** Compute the post-challenge leakage $L_2' := g_2'(L_1, L_2, c_1, \mathsf{sk}_2)$.
- **Challenge 2:**
  1. Compute $(\widetilde{c}_2^1) \leftarrow \mathsf{Encap}_2^*(\mathsf{pk}_2)$.
  2. Compute $\widetilde{k}_2 = \mathsf{Decap}(\mathsf{sk}_2, \widetilde{c}_2^1)$.
  3. Compute $c_2^2 := m_b \oplus \widetilde{k}_2$. Set $c_2 := (\widetilde{c}_2^1, c_2^2)$.
- **Post-challenge Leakage 2:** Let, for a fixed public key pk_1, the random variable $\mathcal{SK}_1$ denote the distribution over the secret keys corresponding to pk_1 from which the key generation algorithm Gen samples the secret key sk_1. Also, consider the following function $f(\mathcal{SK}_1) := (g_1(\mathcal{SK}_1), \mathsf{Decap}(c_1, \mathcal{SK}_1))$. Sample a new secret key $\widetilde{\mathsf{sk}_1}$ from $\mathcal{SK}_1 | f(\mathcal{SK}_1) = (L_1, U_{\mathcal{K}})$.

  Compute the post-challenge leakage $L_1' := g_1'(L_1, L_2, c_1, \widetilde{\mathsf{sk}_1})$. Output L_1' and L_2'.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (\tilde{S}^1, S^2))$

Claim 6.7. The adversarial view in hybrid H_6 is identical to its view in hybrid H_5.

Proof. Note that in order to compute L_2', only $L_1, L_2, c_1, \mathsf{sk}_2$ are required. All of these values are available after the challenge 1 stage. This indicates that we can compute the post-challenge leakage L_2' on the secret key sk_2 given only $L_1, L_2, c_1, \mathsf{sk}_2$, but *without seeing* c_2.

Hybrid H_7: This experiment differs from H_6 in only one step where we replace the key $\widetilde{k}_2$ with a uniform random key $\widehat{k}_2$.

- **Challenge 2:**
  1. Compute $(\widetilde{c}_2^1) \leftarrow \mathsf{Encap}_2^*(\mathsf{pk}_2)$.
  2. Sample $\widehat{k}_2 \leftarrow U_{\mathcal{K}}$.
  3. Compute $\widetilde{c}_2^2 := m_b \oplus \widehat{k}_2$. Set $c_2 := (\widetilde{c}_2^1, \widetilde{c}_2^2)$.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (\tilde{S}^1, S^2))$

Claim 6.8. The adversarial view in hybrid H_7 is statistically indistinguishable from its view in hybrid H_6.

Proof. Since, as noted above, we compute the post-challenge leakage L_2' on the secret key sk_2 given only $L_1, L_2, c_1, \mathsf{sk}_2$, but *without seeing* c_2, indistinguishability follows from the leakage smoothness property of the IB-HPS $\mathcal{P}_2$.

Hybrid H_8: This experiment differs from H_7 in only one step where we replace the ciphertext with a uniform random value.

- **Challenge 2:**
  1. Compute $(\widetilde{c}_2^1) \leftarrow \mathsf{Encap}_2^*(\mathsf{pk}_2)$.
  2. $\underline{\text{Sample } \widetilde{c}_2^2 \leftarrow U.}$ Set $c_2 := (\widetilde{c}_2^1, \widetilde{c}_2^2)$.
- The view of adversary consists of $(\mathsf{pk}_1, \mathsf{pk}_2, L_1, L_2, c_1, L_1', L_2', (\tilde{S}^1, S^2))$

Claim 6.9. The adversarial view in hybrid H_8 is identical to its view in hybrid H_7.

Proof. The proof for this claim is trivial (and similar to that of claim 6.6).

Claim 6.10. The advantage of the adversary in hybrid H_8 is 0.

Proof. This immediately follows since the adversarial view in hybrid H_7 is independent of the challenge bit b.

This completes the proof of theorem 2.

6.2 Chosen Ciphertext Attack (CCA) Security for SS-BRM-PKE

The adversary $\mathcal{A}$ is given access to a decryption oracle Dec, which provides decryption of arbitrary ciphertext along with the decryption access patterns. Note that, after the challenge ciphertext c is given to $\mathcal{A}$, it has access to a restricted decryption oracle $\mathsf{Dec}_{\neg c}$, which provides decryption of arbitrary ciphertext along with the decryption access patterns, except for the challenge ciphertext c.

- **Key Generation:** The challenger computes $(\mathsf{pk}, \mathsf{sk}_1||\mathsf{sk}_2) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^\ell)$ and gives pk to the adversary $\mathcal{A}$.
- **Message Commitment:** The adversary $\mathcal{A}$ selects two messages m_0, m_1.
- **Pre-challenge Leakage:** The adversary $\mathcal{A}$ selects a PPT function $g := (g_1, g_2) : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^\ell \times \{0,1\}^\ell$ and gets $L_1 := g_1(\mathsf{sk}_1)$, $L_2 := g_2(\mathsf{sk}_2)$ from the challenger.
- **Pre-challenge Decryption Queries:** The adversary $\mathcal{A}$ queries the decryption oracle Dec to obtain the decryption of arbitrary ciphertext of its choice along with the decryption access patterns. $\mathcal{A}$ can query Dec $\text{poly}(\lambda, \ell)$ times.
- **Challenge:** The challenger chooses $b \leftarrow \{0,1\}$ uniformly at random and gives $c \leftarrow \mathsf{Encrypt}(m_b, \mathsf{pk})$ to the adversary $\mathcal{A}$.
- **Encryption Access Patterns:** The challenger also sends the access pattern $(i^{(1)}, i^{(2)}, \ldots, i^{(t)})$ corresponding to the encryption procedure, to $\mathcal{A}$.
- **Post-challenge Leakage:** The adversary $\mathcal{A}$ selects a PPT function $g' := (g_1', g_2') : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^\ell \times \{0,1\}^\ell$ and gets $L_1' := g_1'(L_1, L_2, c, \mathsf{sk}_1)$, $L_2' := g_2'(L_1, L_2, c, \mathsf{sk}_2)$ from the challenger.
- **Post-challenge Decryption Queries:** The adversary $\mathcal{A}$ queries the decryption oracle $\mathsf{Dec}_{\neg c}$ to obtain the decryption of arbitrary ciphertext of its choice along with the decryption access patterns. $\mathcal{A}$ can query $\mathsf{Dec}_{\neg c}$ $\text{poly}(\lambda, \ell)$ times.
- **Decryption Access Patterns:** The challenger also sends the access pattern $(S^1, S^2) \xleftarrow{Access} \mathsf{Decrypt}^{sk_1||\mathsf{sk}_2}(c)$, to $\mathcal{A}$. S^i is a set of indices s_j^i of sk_i for $i \in \{1,2\}$ and $j \in [n]$, where $|\mathsf{sk}_1| = |\mathsf{sk}_2| = n$. Also, $|S^i| = t$, where t is the number of locations of sk_1 and sk_2 required to be accessed to decrypt any ciphertext.
- **Output:** The adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$. We say that $\mathcal{A}$ wins the game if $b' = b$.

For any adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is defined as $\mathbf{Adv}_{\mathcal{E},\mathcal{A}}^{\mathsf{SS\text{-}BRM\text{-}PKE-CCA}}(\lambda, \ell) \overset{\text{def}}{=} |\Pr[\mathcal{A}\,\text{wins}] - \frac{1}{2}|$. It should be noted that the decryption access patterns indicating which parts of secret key were accessed during the decryption must be provided to the adversary only *after* the leakage information, otherwise the adversary can simply ask for the leakage on the secret key positions “relevant” to the decryption of the challenge ciphertext.

6.3 Chosen Ciphertext Attack (CCA) Secure Construction for SS-BRM-PKE

In this section, we explain how to construct a CCA-secure SS-BRM-PKE scheme from a given CPA-secure SS-BRM-PKE scheme. This is achieved using the well known double encryption paradigm introduced by Naor-Yung [61], in fact we use a simpler construction presented in [65,55,8].

The high level idea here is to encrypt the plaintext twice with independent keys obtained from CPA-secure scheme and then attach a NIZK proof ensuring that both the ciphertexts are indeed encryptions of the same plaintext. This scheme is CCA-1 secure, however even CCA-2 security can be achieved if the NIZK proof system is one-time simulation-sound [65]. The intuition behind the proof is that under the promise that the two ciphertexts encrypt the same plaintext (where the promise is ensured by the soundness of the NIZK proofs), the behavior of the decryption oracle is identical whether decrypting under the first or second CPA key. This allows us to switch the underlying plaintext of the second CPA ciphertext in the challenge ciphertext, while decrypting under the first key and subsequently to switch the underlying plaintext of the first CPA ciphertext in the challenge ciphertext, while decrypting under the second key. We now present the construction and proof, note that this is same as presented in [55], and is presented here only for the sake of completeness.

Construction $\mathcal{E}_C = (\mathsf{KeyGen}_C, \mathsf{Encrypt}_C, \mathsf{Decrypt}_C)$ Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a CPA-secure SS-BRM PKE scheme, and $\mathit{\Pi} = (\mathcal{P}, \mathcal{V})$ be a one-time simulation-sound adaptive NIZK proof system.

- Key Generation (KeyGen_C): Generate two independent sets of key from KeyGen as $(\mathsf{pk}_1, \mathsf{sk}_{11}||\mathsf{sk}_{12}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda, 1^\ell)$ and $(\mathsf{pk}_2, \mathsf{sk}_{21}||\mathsf{sk}_{22}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda, 1^\ell)$. Also, choose the uniform reference string crs for the NIZK scheme $\mathit{\Pi}$. Set the public key as $\mathsf{pk}_C = (\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{crs})$, and the secret as $\mathsf{sk}_C = (\mathsf{sk}_C^1||\mathsf{sk}_C^2) = ((\mathsf{sk}_{11}, \mathsf{sk}_{21})||(\mathsf{sk}_{12}, \mathsf{sk}_{22}))$.
- Encryption ($\mathsf{Encrypt}_C(m)$): Encrypt the plaintext m as follows, compute $c_1 = \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}_1}(m)$ and $c_2 = \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}_2}(m)$. Then obtain the NIZK proof $\pi = \mathit{\Pi}.\mathcal{P}(c_1, c_2, \mathsf{pk}_1, \mathsf{pk}_2)$ (note that m is witness for π and $\mathit{\Pi}.\mathcal{V}(\pi, m) = 1$ can be computed efficiently). Output $\mathsf{Encrypt}_C(m) = (c_1, c_2, \pi)$.
- Decryption ($\mathsf{Decrypt}_C(c_1, c_2, \pi)$: First verify the validity of the proof, by checking if $\mathit{\Pi}.\mathcal{V}(\pi, m) = 1$. If the proof is accepted, then decrypt c_1 (or c_2) to obtain $m = \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}_{11}||\mathsf{sk}_{12}}(c_1)$ (or $m = \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}_{21}||\mathsf{sk}_{22}}(c_2)$). Output m.

Theorem 3. *If* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *be a* CPA-*secure* SS-BRM-PKE *scheme, and* $\mathit{\Pi} = (\mathcal{P}, \mathcal{V})$ *be a one-time simulation-sound adaptive NIZK proof system. Then,* $\mathcal{E}_C = (\mathsf{KeyGen}_C, \mathsf{Encrypt}_C, \mathsf{Decrypt}_C)$ *is a* CCA-2 *secure* SS-BRM-PKE *scheme.*

Informal Proof (for Theorem 3): In this section, we present an informal proof for security of generic transformation used to obtain CCA-2 security from CPA secure encryption schemes using the double encryption paradigm described above. Note that the detailed proof for Theorem 3 follows exactly the same hybrids as presented in [55]. For each two consecutive hybrids, [55] presents a reduction to prove indistinguishability of the hybrids. Note that in the case where such reduction honestly generates the two secret keys for the underlying CPA secure encryption scheme itself, simulating the leakage required by the SS-BRM-PKE setting is trivial. Thus, we focus only on the reductions to the CPA security of the underlying scheme, since this is the only case where the reduction does not know both secret keys.

In the reduction to CPA security of the underlying encryption scheme, it is shown that an adversary $\mathcal{A}_{\mathsf{CPA}}$ who only has access to the encryption oracle, can simulate the role of the decryption oracle for an adversary $\mathcal{A}_{\mathsf{CCA2}}$ trying to break the security of $\mathcal{E}_C$. Essentially the same reduction is used twice, once when the CPA challenge ciphertext is embedded in the first position of the CCA-2 challenge ciphertext and once when the it is embedded in the second position. We present here only the case where the CPA challenge ciphertext is embedded in the first position of the CCA-2 challenge ciphertext, since the other case is symmetric.

Note that in order to decrypt a given ciphertext, *only one* of the secret keys (either $(\mathsf{sk}_{11}||\mathsf{sk}_{12})$ or $(\mathsf{sk}_{21}||\mathsf{sk}_{22})$) is required. The following steps show how $\mathcal{A}_{\mathsf{CPA}}$ can be used to simulate the decryption oracle:

- Initially, $\mathcal{A}_{\mathsf{CPA}}$ receives pk_1 from its challenger. $\mathcal{A}_{\mathsf{CPA}}$ then generates $(\mathsf{pk}_2, \mathsf{sk}_{21}||\mathsf{sk}_{22}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda, 1^\ell)$ on its own. $\mathcal{A}_{\mathsf{CPA}}$ also chooses the uniform reference string crs for the NIZK scheme Π. $\mathcal{A}_{\mathsf{CPA}}$ publishes public key as $\mathsf{pk}_C = (\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{crs})$. Note that since $\mathcal{A}_{\mathsf{CPA}}$ knows $\mathsf{sk}_{21}||\mathsf{sk}_{22}$; it can decrypt any ciphertext computed using pk_2. Therefore, $\mathcal{A}_{\mathsf{CPA}}$ can decrypt any ciphertext submitted by $\mathcal{A}_{\mathsf{CCA2}}$.
- Next, $\mathcal{A}_{\mathsf{CPA}}$ receives the challenge ciphertext c_1 from its challenger and prepares the challenge ciphertext for $\mathcal{A}_{\mathsf{CCA2}}$. $\mathcal{A}_{\mathsf{CPA}}$ computes $c_2 = \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}_2}(0)$. $\mathcal{A}_{\mathsf{CPA}}$ then simulates the proof π using the one-time simulation soundness property of NIZK scheme Π. $\mathcal{A}_{\mathsf{CPA}}$ then submits $c = (c_1, c_2, \pi)$ to $\mathcal{A}_{\mathsf{CCA2}}$.
- Finally, $\mathcal{A}_{\mathsf{CPA}}$ outputs the same output as $\mathcal{A}_{\mathsf{CCA2}}$. This is useful, since any information learned by $\mathcal{A}_{\mathsf{CCA2}}$ about $c = (c_1, c_2, \pi)$ can be used by $\mathcal{A}_{\mathsf{CPA}}$ to distinguish c_1.

The only difference in our setting is simulation of leakage queries. It is easy to see that $\mathcal{A}_{\mathsf{CPA}}$ can compute the leakage on $(\mathsf{sk}_{21}||\mathsf{sk}_{22})$ since it knows $(\mathsf{sk}_{21}||\mathsf{sk}_{22})$. In order to simulate leakage on $(\mathsf{sk}_{11}||\mathsf{sk}_{12})$, it can simply forward the leakage queries to its own challenger and receive the leakage.

7 Our Construction

We first present our construction, for achieving resilience against one time leakage and partial tampering attacks in Section 7.1; and then extend it to achieve full resilience against one time leakage and tampering attacks in Section 7.2.

7.1 Partial One-Time Tamper Leakage Resilience (OT-TLR)

Definition 11 (Collision-Resistant Hash Function Family). *[31] A family of hash functions $\mathcal{H} := \{h : \{0,1\}^* \to \{0,1\}^m\}$ is (t, δ)-collision-resistant if for any (non-uniform) attacker $\mathcal{B}$ of size at most t,*

$$\Pr[H(X_1) = H(X_2) \wedge X_1 \neq X_2] \leq \delta \text{ where } H \leftarrow \mathcal{H} \text{ and } (X_1, X_2) \leftarrow \mathcal{B}(H).$$

Definition 12 (Merkle Tree). *Let $h : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$ be a hash function that maps two blocks of messages to one.[3] A Merkle Tree $\mathsf{Tree}_h(M)$ takes as input a message $M = (m_1, m_2, \ldots, m_n) \in \mathcal{X}^n$. Then it applies the hash on each pair (m_{2i-1}, m_{2i}), resulting in $n/2$ blocks. Then again, it partitions the blocks into pairs and applies the hash on the pairs, which results in $n/4$ blocks. This is repeated $\log n$ times, resulting a binary tree with hash values, until one block remains. We call this value the root of Merkle Tree denoted $\mathsf{Root}_h(M)$, and the internal nodes (including the root) as $\mathsf{Tree}_h(M)$. Here M can be viewed as leaves.*

Theorem 4. *Assuming h is a collision resistant hash function. Then for any message $M = (m_1, m_2, \ldots, m_n) \in \mathcal{X}^n$, any polynomial time adversary $\mathcal{A}$, $\Pr\Big[(m_i', p_i) \leftarrow \mathcal{A}(M, h) : m_i' \neq m_i, p_i$ is a consistent path with $\mathsf{Root}_h(M)\Big] \leq \mathsf{negl}(\lambda)$.*

Moreover, given a path p_i passing through the leaf m_i; and a new value m_i', there is an algorithm that computes $\mathsf{Root}_h(M')$ in time $\mathrm{poly}(\log n, \lambda)$, where $M' = (m_1, \ldots, m_{i-1}, m_i', m_{i+1}, \ldots, m_n)$.

Definition 13 (Entropy). *The min-entropy of a random variable X is defined as $H_\infty(X) = -\log(\max_x \Pr[X = x])$. The conditional min-entropy of a random variable X, conditioned on the experiment $\mathcal{E}$ is $\tilde{H}_\infty(X|\mathcal{E}) = -\log(\max_{\mathcal{A}} \Pr\big[\mathcal{A}^{\mathcal{E}(\cdot)}() = X\big])$. In the special case that $\mathcal{E}$ is a non-interactive experiment which simply outputs a random variable Z, it is written as $\tilde{H}_\infty(X|Z)$.*

Definition 14 (Seed-Dependent Condenser). *[31] An efficient function $\mathsf{Cond} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is a seed-dependent $([\boldsymbol{H}_\infty \geq k] \to_\varepsilon [\boldsymbol{H}_\infty \geq k'], t)$-condenser if for all probabilistic adversaries $\mathcal{A}$ of size at most t who take a random seed $S \leftarrow \{0,1\}^d$ and output (using more coins) a sample $X \leftarrow \mathcal{A}(S)$ of entropy $\boldsymbol{H}_\infty(X|S) \geq k$, the joint distribution $(S, \mathsf{Cond}(X; S))$ is ε-close to some (S, R), where $\boldsymbol{H}_\infty(R|S) \geq k'$.*

[3] Here we assume $|\mathcal{X}|$ is greater than the security parameter.

Theorem 5. *[31] Fix any $\beta > 0$. If $\mathcal{H}$ is a $(2t, 2^{\beta-1}/2^m)$-collision-resistant hash function family, then $\mathsf{Cond}(X; H) \overset{def}{=} H(x)$ for $H \leftarrow \mathcal{H}$ is a seed-dependent $(([\mathbf{H}_\infty \geq m - \beta + 1] \rightarrow [\mathbf{H}_\infty \geq m - \beta + \log \varepsilon]), t)$-condenser.*

We next present our construction:

Construction Π = $(\mathsf{ENC}, \mathsf{DEC}, \mathsf{UPDATE})$. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a CCA-secure public-key encryption scheme in the BRM, $\mathcal{V} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme in the BRM and H is a family of collision resistance hash functions. Then we consider the following coding scheme:

- $\mathsf{ENC}(D)$: On input database $D = (D_1, D_2, \ldots, D_n) \in \Sigma^n$:
  - Choose $(\mathsf{pk}, \mathsf{sk}_\varepsilon) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda, 1^\ell)$, $(\mathsf{vk}, \mathsf{sk}_\sigma) \leftarrow \mathcal{V}.\mathsf{Gen}(1^\lambda)$ and $h \leftarrow H$.
  - Set $\tilde{D}_0 := 0$.
  - Compute $\tilde{D}_i \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(D_i)$ for $i \in [n]$. Let $\tilde{D} := \tilde{D}_0, \ldots, \tilde{D}_n$. Set[4] $T := \mathsf{Tree}_h(\tilde{D})$, $\sigma := \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R)$, where $R := \mathsf{Root}_h(\tilde{D})$.
  - Output codeword $C := (C_1, C_2)$, where $C_1 := (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h)$ and $C_2 := (\tilde{D}, T, R, \sigma)$. We assume $C_1 \in \hat{\Sigma}^{n_1}, C_2 \in \hat{\Sigma}^{n_2}$.
- $\mathsf{DEC}^C(i)$: On input $i \in [n]$:
  - Parse $C := (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h, \tilde{D}, T, R, \sigma)$.
  - Check whether $\tilde{D}_0 := 0$. If not, output $\perp$ and terminate.
  - Read path p_i in T, corresponding to leaf i and use p_i to recompute $\hat{R} = \mathsf{Root}_h(p_i)$.
  - Check that $\hat{R} := R$. If not, output $\perp$ and terminate.
  - Check that $\mathcal{V}.\mathsf{Verify}(\mathsf{vk}, R, \sigma) = 1$. If not, output $\perp$ and terminate.
  - Output $D_i := \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}_\varepsilon}(\tilde{D}_i)$.
- $\mathsf{UPDATE}^C(i, v)$: On inputs an index $i \in [n]$, a value $v \in \Sigma$:
  - Run $\mathsf{DEC}^C(i)$. If it outputs $\perp$, set $\tilde{D}_0 := 1$, write back to memory and terminate.
  - Parse $C := (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h, \tilde{D}, T, R, \sigma)$.
  - Compute $\tilde{D}'_i \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{sk}_\varepsilon}(v)$. Let $\tilde{D}' := \tilde{D}_0, \ldots, \tilde{D}_{i-1}, \tilde{D}'_i, \tilde{D}_{i+1}, \ldots, \tilde{D}_n$.
  - Read path p_i in T, corresponding to leaf i and use $(p_i, \tilde{D}'_i)$ to compute a new path p'_i (that replaces $\tilde{D}_i$ by $\tilde{D}'_i$). Set $R' = \mathsf{Root}_h(p'_i)$. Let T' denote the updated tree.
  - Compute $\sigma' := \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R')$.
  - Write back $(\tilde{D}'_i, p'_i, R', \sigma')$ yielding updated codeword $C' := (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h, \tilde{D}', T', R', \sigma')$.

Locality of the construction. Using the BRM scheme of [8] (along with sub-exponentially hard PRG), we have that DEC and UPDATE make polylog(λ) number of random accesses to C.

Theorem 6. *For security parameter $\lambda \in \mathbb{N}$, leakage parameter $\ell := \ell(\lambda)$, alphabet Σ such that $\log |\Sigma| \in \Omega(\lambda)$, and database size $n := n(\lambda)$: Assume $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is a CCA-secure public-key encryption scheme in the BRM with leakage parameter ℓ and relative leakage $\alpha < 1$, $\mathcal{V} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a signature scheme in the BRM with leakage parameter ℓ and relative leakage $\alpha < 1$, and H is a family of collision resistance hash functions with sub-exponential security. Then Π is a one-time tamper and leakage resilient locally decodable and updatable code taking messages in Σ^n to codewords in $\hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2}$, which is secure against tampering class*

$$\bar{\mathcal{F}} \overset{\text{def}}{=} \left\{ \begin{array}{l} f : \Sigma^{n_1} \times \hat{\Sigma}^{n_2} \rightarrow \hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2} \text{ and } |f| \leq \mathrm{poly}(\lambda), \text{ such that :} \\ f = (f_1, f_2),\ f_1 : \hat{\Sigma}^{n_1} \rightarrow \hat{\Sigma}^{n_1},\ f_2 : \hat{\Sigma}^{n_2} \rightarrow \hat{\Sigma}^{n_2} \text{ and } f_1 \text{ is the identity function.} \end{array} \right\},$$

and is leakage resilient against the class

$$\bar{\mathcal{G}} \overset{\text{def}}{=} \left\{ \begin{array}{l} g : \hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2} \rightarrow \{0,1\}^\ell \times \{0,1\}^{\frac{n_2 \cdot \log |\hat{\Sigma}|}{5}} \text{ and } |g| \leq \mathrm{poly}(\lambda), \text{ such that :} \\ g = (g_1, g_2),\ g_1 : \hat{\Sigma}^{n_1} \rightarrow \{0,1\}^\ell,\ g_2 : \hat{\Sigma}^{n_2} \rightarrow \{0,1\}^{\frac{n_2 \cdot \log |\hat{\Sigma}|}{5}}. \end{array} \right\}.$$

[4] We additionally pad the tree T with dummy leaves consisting of uniform random values to ensure that the relative leakage on the second split state, C_2 is at least $\frac{1}{5}$. See the proof of Claim 7.4 for details.

Moreover, Π has relative leakage $\min(\frac{\alpha}{2}, \frac{1}{5})$.

Remark 2. As discussed in Remark 1, the primitive we require is slightly weaker than entropic signatures in the BRM and can be constructed in the standard model, without use of random oracles. We next sketch the relaxation in our setting, and show how to construct BRM signatures in our setting. First, note that if we care only about locality of verification, but not signing, we can construct BRM signatures by taking a signature scheme in the relative leakage model (with additional properties–see discussion in [13]) such as [52], generating n verification key/signing key pairs and having a signature consist of n signatures under all n pairs. To verify, choose a random subset of t signatures and verify them. The resulting verification key can be shortened using techniques of [9]. To achieve a construction with a short signature, the signer should instead sign the message only under a random subset of t signatures. But if the signer chooses the subset of keys itself, then forging is trivial by leaking the corresponding t keys ahead of time. Alwen et al. [9] solved this problem by using the hash of the message $H(m)$ (where H is a RO) to select the random subset. In our setting, we can use a signature scheme with a long signature a *single* time, during the *encode* procedure. During encode, we choose a random subset $\mathcal{T} \subseteq [n]$ of size $|\mathcal{T}| = t$ and sign (a representation of) the set $\mathcal{T}$ using the non-local scheme described above. When a new signature is later computed, the message is signed only under the secret keys corresponding to the random subset $\mathcal{T}$. To verify a signature, all t positions of the second signature are verified and then the set $\mathcal{T}$ is verified by choosing a random subset of t positions in the first (long) signature and checking them. Note that an attacker cannot forge the first (long) signature, since it would require leaking almost the entire signature key. On the other hand, since the attacker does not know $\mathcal{T}$ before leaking on the signing key, the properties of [52] ensure that the attacker cannot forge the second signature either.

Proof (of Theorem 6). To prove the theorem, for any efficient adversary $\mathcal{A}$, we need to construct a simulator $\mathcal{S}$, such that for any initial database $D \in \Sigma^n$ and any efficient updater $\mathcal{U}$, the experiment of one-time attacks $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D}$ is indistinguishable from the ideal experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},D}$.

The simulator $\mathcal{S}$ first samples random coins for the updater $\mathcal{U}$, so its output just depends on its input given the random coins. Then $\mathcal{S}$ works as follows:

- Initially $\mathcal{S}$ samples $(\mathsf{pk}, \mathsf{sk}_\varepsilon) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda, 1^\ell), (\mathsf{vk}, \mathsf{sk}_\sigma) \leftarrow \mathcal{V}.\mathsf{Gen}(1^\lambda), h \leftarrow H$, sets $\tilde{D}_0 = 0$ and then generates n encryptions of 0, i.e., $\tilde{D}_1, \tilde{D}_2, \ldots, \tilde{D}_n$ where $\tilde{D}_i \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(0)$ for $i \in [n]$. Let $\tilde{D}^{(1)} := \tilde{D}_0, \tilde{D}_1, \ldots, \tilde{D}_n$. Then $\mathcal{S}$ computes $T^{(1)} := \mathsf{Tree}_h(\tilde{D}^{(1)})$. Let $\sigma^{(1)} = \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R^{(1)})$, where $R^{(1)}$ is the root of the tree. $\mathcal{S}$ keeps global variables $\mathsf{flag}, \mathsf{Leaked}, \mathsf{Tampered} = \mathsf{0}$.
- At each round j, let $C^{(j)} := (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h, \tilde{D}^{(j)}, T^{(j)}, R^{(j)}, \sigma^{(j)})$ denote the current simulated codeword stored by $\mathcal{S}$ and let $\boldsymbol{w}^{(j)}$ denote the simulator's output in the previous round. In the first round, $w_i^{(0)} :=$ *same* for all $i \in [n]$. In each round, $\mathcal{S}$ does the following:

  Simulating Update:
  - If $\mathsf{flag} = \mathsf{0}$, $\mathcal{S}$ does the following: Receives an index $i^{(j)} \in [n]$ from the updater. Runs $\mathsf{UPDATE}^{C^{(j)}}(i^{(j)}, 0)$. Let $C^{(j+1)}$ be the resulting codeword after the update.
  - If $\mathsf{flag} = \mathsf{1}$, $\mathcal{S}$ does the following: Computes $(i^{(j)}, v) \leftarrow \mathcal{U}(\boldsymbol{w}^{(j)})$ on his own, and runs $\mathsf{UPDATE}^{C^{(j)}}(i^{(j)}, v)$. Let $C^{(j+1)}$ be the resulting codeword after the update.

  Simulating the Round's Output:
  - $\mathcal{S}$ sets $(\tilde{D}_0, \tilde{D}_1, \ldots, \tilde{D}_n) := \tilde{D}^{(j+1)}$, $T := T^{(j+1)}$, and $R := R^{(j+1)}$ and $\sigma = \sigma^{(j+1)}$.
  - $\mathcal{S}$ emulates the adversary $\mathcal{A}$ and receives some $(g_1, g_2) \in \bar{\mathcal{G}}, (f_1, f_2) \in \bar{\mathcal{F}}$ (such that f_1 is the identity function).
  - If Leaked is 0, then $\mathcal{S}$ computes $\ell_1 := g_1(\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h)$, $\ell_2 := g_2(\tilde{D}^{(j+1)}, T^{(j+1)}, R^{(j+1)}, \sigma^{(j)})$, sets $\ell := (\ell_1, \ell_2)$ and sets Leaked to 1.
  - If Leaked is 1 and $\mathsf{Tampered}$ is 0, then $\mathcal{S}$ computes

$$\begin{aligned} C' &:= (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h, \tilde{D}' := (\tilde{D}'_0, \tilde{D}'_1, \ldots, \tilde{D}'_n), T', R', \sigma') \\ &:= (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h, f_2(\tilde{D}^{(j+1)}, T^{(j+1)}, R^{(j+1)}, \sigma^{(j+1)})) \end{aligned}$$

and sets $\mathsf{Tampered}$ to 1.

- If $\mathsf{flag} = 0$, $\mathcal{S}$ does the following:
  - $\mathcal{S}$ sets $\mathcal{I}^{(j+1)} = \{u \in [n] : \tilde{D}'_u \neq \tilde{D}_u \vee \mathsf{DEC}^{C'}(u) = \perp\}$, i.e. the indices where $\tilde{D}'$ is not equal to $\tilde{D}$ or where decode evaluates to $\perp$. If $\mathcal{I}^{(j+1)} = [n]$, $\mathcal{S}$ sets $\mathsf{flag} := 1$. If $\mathcal{I}^{(j+1)} \neq [n]$, $\mathcal{S}$ outputs $\{\ell, \boldsymbol{w}^{(j+1)}\}$, where $w^{(j+1)}[i] = \perp$ for $i \in \mathcal{I}^{(j+1)}$ and $w^{(j+1)}[i] = \textit{same}$ for $i \notin \mathcal{I}^{(j+1)}$.
- If $\mathsf{flag} = 1$, $\mathcal{S}$ simulates the real experiment faithfully: For $i \in [n]$, $\mathcal{S}$ sets $\boldsymbol{w}^{(j+1)}[i] := \mathsf{DEC}^{(C')}(i)$, i.e. running the real decoding algorithm. Then $\mathcal{S}$ outputs $\{\ell, \boldsymbol{w}^{(j+1)}\}$.

To show $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D} \approx \mathbf{Ideal}_{\mathcal{S},\mathcal{U},D}$, we consider several intermediate hybrids.

Hybrid H_0: This is exactly the experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$.

Hybrid H_1: This experiment is the same as H_1 except the simulator does not encrypt all 0's (i.e. $\mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(0)$); instead, it encrypts the real messages (in the first place, and in the update when $\mathsf{flag} = 0$), as in the real experiment.

Claim 7.1. $H_0 \overset{c}{\approx} H_1$.

Proof. The claim is due to reduction argument to the CCA security of the underlying encryption scheme in the BRM model.

We next consider the probability of two events occurring in Hybrid H_1:

Event EV_1: At some point during the simulator's execution, we have that all of the following hold:

- $\mathcal{I}^{(j+1)} \neq [n]$.
- $R' \neq R^{(j+1)}$.
- $\mathsf{Verify}(\mathsf{vk}, R', \sigma') = 1$.

Claim 7.2. For every initial database D, updater $\mathcal{U}$, and adversary $\mathcal{A}$, EV_1 occurs with negligible probability in H_1.

Proof. Assume towards contradiction that for initial database D, updater $\mathcal{U}$, and adversary $\mathcal{A}'$, EV_1 occurs with non-negligible probability in H_1. Then there is some round j^* at which EV_1 occurs with non-negligible probability. We will construct an entropic adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ whose advantage in the game $EUG_\ell^\lambda(\mathcal{V}, \mathcal{A})$ is non-negligible. This leads to contradiction of the security of the signature scheme $\mathcal{V}$.

Definition of $\mathcal{A}_1$: $\mathcal{A}_1$ receives vk and oracle access to $\mathcal{S}^{\mathsf{sk}_\sigma}(\cdot)$ and the leakage oracle $\mathcal{O}^{\lambda,\ell}_{\mathsf{sk}_\sigma}(\cdot)$ from the external game and simulates the leakage viewed by the adversary on sk_σ in the following way:

- $\mathcal{A}_1$ samples $(\mathsf{pk}, \mathsf{sk}_\varepsilon) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda, 1^\ell), h \leftarrow H$.
- $\mathcal{A}_1$ instantiates $\mathcal{A}'$ and waits for its leakage query. Finally, $\mathcal{A}_1$ receives some $(g_1, g_2) \in \bar{\mathcal{G}}, (f_1, f_2) \in \bar{\mathcal{F}}$ from $\mathcal{A}'$.
- $\mathcal{A}_1$ queries its leakage oracle to compute $\ell_1 := g_1(\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, \mathsf{sk}_\sigma, h)$ and outputs $(\ell_1, \mathsf{pk}, \mathsf{sk}_\varepsilon, h)$ to $\mathcal{A}_2$.

Definition of $\mathcal{A}_2$: $\mathcal{A}_2$ receives vk, hint $(\ell_1, \mathsf{pk}, \mathsf{sk}_\varepsilon, h)$ from $\mathcal{A}_1$ and oracle access to $\mathcal{S}^{\mathsf{sk}_\sigma}(\cdot)$ from the external game and instantiates the simulator in H_1 up to round j^* with the following changes:

- Initially $\mathcal{A}_2$ sets $\tilde{D}_0 = 0$ and then generates n encryptions, $\tilde{D}_1, \tilde{D}_2, \ldots, \tilde{D}_n$ where $\tilde{D}_i \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(D_i)$ for $i \in [n]$. Let $\tilde{D}^{(1)} := \tilde{D}_0, \tilde{D}_1, \ldots, \tilde{D}_n$. Then $\mathcal{A}_2$ computes $T^{(1)} := \mathsf{Tree}_h(\tilde{D}^{(1)})$. $\mathcal{A}_2$ queries its oracle to obtain $\sigma^{(1)} = \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R^{(1)})$, where $R^{(1)}$ is the root of the tree $T^{(1)}$. $\mathcal{A}_2$ keeps global variables $\mathsf{flag}, \mathsf{Leaked}, \mathsf{Tampered} = 0$.
- At each round $j \leq j^*$, let $C^{(j)} := (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, *, h, \tilde{D}^{(j)}, T^{(j)}, R^{(j)}, \sigma^{(j)})$ denote the current simulated codeword stored by $\mathcal{A}_2$ and let $w^{(j)}$ denote the simulator's output in the previous round. In the first round, $\boldsymbol{w}_i^{(0)} := D_i$ for all $i \in [n]$. In each round, $\mathcal{A}_2$ does the following:

Simulating Update:

– Compute $(i^{(j)}, v) \leftarrow \mathcal{U}(\boldsymbol{w}^{(j)})$ on its own, and runs $\mathsf{UPDATE}^{C^{(j)}}(i^{(j)}, v)$. When a signature is required, $\mathcal{A}_2$ uses its signing oracle to compute the signature. Let $C^{(j+1)}$ be the resulting codeword after the update.

Simulating the Round's Output:

– $\mathcal{A}_2$ sets $(\tilde{D}_0, \tilde{D}_1, \ldots, \tilde{D}_n) := \tilde{D}^{(j+1)}$, $T := T^{(j+1)}$, and $R := R^{(j+1)}$ and $\sigma = \sigma^{(j+1)}$.

– $\mathcal{A}_2$ emulates the adversary $\mathcal{A}'$ and receives some $(g_1, g_2) \in \bar{\mathcal{G}}$, $(f_1, f_2) \in \bar{\mathcal{F}}$.

– If Leaked is 0, then $\mathcal{A}_2$ computes $\ell_2 := g_2(\tilde{D}^{(j+1)}, T^{(j+1)}, R^{(j+1)}, \sigma^{(j)})$, sets $\ell := (\ell_1, \ell_2)$, outputs ℓ to $\mathcal{A}'$ and sets Leaked to 1.

– If Leaked is 1 and $\mathsf{Tampered}$ is 0, then $\mathcal{A}_2$ computes

$$\begin{aligned} C' &:= (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, *, h, \tilde{D}' := (\tilde{D}'_0, \tilde{D}'_1, \ldots, \tilde{D}'_n), T', R', \sigma') \\ &:= (\mathsf{pk}, \mathsf{sk}_\varepsilon, \mathsf{vk}, *, h, f_2(\tilde{D}^{(j+1)}, T^{(j+1)}, R^{(j+1)}, \sigma^{(j+1)})) \end{aligned}$$

and sets $\mathsf{Tampered}$ to 1.

– $\mathcal{A}_2$ sets $\mathcal{I}^{(j+1)} = \{u \in [n] : \tilde{D}'_u \neq \tilde{D}_u \vee \mathsf{DEC}^{C'}(u) = \perp\}$, i.e. the indices where $\tilde{D}'$ is not equal to $\tilde{D}$ or decode outputs $\perp$. If $\mathcal{I}^{(j+1)} = [n]$, $\mathcal{A}_2$ outputs $(R' \leftarrow \{0,1\}^m, \perp)$ and terminates. If $\mathcal{I}^{(j+1)} \neq [n]$, $\mathcal{A}_2$ computes $\{\ell, \boldsymbol{w}^{(j+1)}\}$, where $w^{(j+1)}[i] = \perp$ for $i \in \mathcal{I}^{(j+1)}$ and $w^{(j+1)}[i] = D^{(j+1)}$ for $i \notin \mathcal{I}^{(j+1)}$, where $D^{(j+1)}$ denotes either the initial value in position i, or the latest value that has been updated.

– If $j = j^*$, $R' \neq R$ and $\mathsf{Verify}(\mathsf{vk}, R', \sigma') = 1$, $\mathcal{A}_2$ outputs $(R' := \mathsf{Root}_h(\tilde{D}'), \sigma')$ and terminates. Otherwise, $\mathcal{A}_2$ outputs $(R' \leftarrow \{0,1\}^m, \perp)$ and terminates.

Claim 7.3. $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is entropic. Specifically,

$$\tilde{H}_\infty(MSG_{\mathcal{A}_2} \mid View_{\mathcal{A}_1}) \geq \lambda,$$

for security parameter λ.

Proof. $\mathcal{A}_2$ is defined such that when event EV_1 occurs it outputs message $R' := \mathsf{Root}_h(\tilde{D}')$ and when event EV_1 does not occur, it outputs a random $R' \in \{0,1\}^m$ (with full entropy). Thus, to lower bound the min-entropy of $\mathcal{A}_2$'s output, it is sufficient to lower bound the min-entropy of $R' := \mathsf{Root}_h(\tilde{D}')$, *conditioned on event EV_1 occurring.*

Towards this goal, recall that the collision-resistant hash function family $H = \{h : \{0,1\}^* \to \{0,1\}^m\}$ is sub-exponential, i.e. the collision probability for $h \in H$ is $\frac{1}{2^{m^{\delta'}}}$ for constant $0 < \delta' < 1$ (where m is the output length of h). This implies that the hash function computed by the Merkle tree, $\mathsf{Root}_h(\cdot)$, which takes as input $\tilde{D}'$ is also sub-exponential, with collision probability $\frac{1}{2^{m^\delta}}$ for constant $0 < \delta < 1$. Thus, Theorem 5 guarantees that $\mathsf{Root}_h(\cdot)$ is a $(([\mathbf{H}_\infty \geq m^\delta] \to [\mathbf{H}_\infty \geq m^\delta + \log \varepsilon]), \mathrm{poly}(\lambda))$-seed dependent condenser. Given δ (which is determined by choice of H) and negligible ε, we set m such that $m^\delta + \log \varepsilon \geq \lambda$, where λ is security parameter.

We must now argue that the input to the condenser, $\tilde{D}'$, has min-entropy at least m^δ when event EV_1 occurs. Indeed, note that when EV_1 occurs, at least one of the encryptions $\tilde{D}'_1, \ldots, \tilde{D}'_n$ is equivalent to one of the "fresh" encryptions generated by $\mathcal{A}_2$ in either the first step or while simulating an update (otherwise, we do not satisfy the condition that $\mathcal{I}^{(j^*+1)} \neq [n]$) and, moreover, is chosen completely independently of the view of $\mathcal{A}_1$. Now, we can trivially modify any CCA-secure public key encryption scheme in the BRM model to a scheme in which ciphertexts contain at least m^δ bits of min-entropy (even for a fixed plaintext), by padding the message with m^δ number of random bits. Thus, we have that, conditioned on EV_1 occurring, $\tilde{D}' := \tilde{D}'_1, \ldots, \tilde{D}'_n$ contains sufficient min entropy, which completes the proof of claim 7.3.

Claim 7.4. Assuming EV_1 occurs with non-negligible probability, $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ successfully forges with non-negligible probability.

Proof. If EV_1 occurs with non-negligible probability in round j^*, it exactly means that (1) $\mathcal{I}^{(j^*+1)} \neq [n]$, (2) $R' \neq R^{(j+1)}$, (3) $\mathsf{Verify}(\mathsf{vk}, R', \sigma') = 1$. Note that if the above occur in round j^*, then the output of $\mathcal{A}_2$ is distributed identically to (R', σ') as above. Moreover, if (2) and (3) occur, it means that either (R', σ') is a replay of a previous root and signature (which were queried previously to the signing oracle) or (R', σ') is indeed a forgery. Due to collision resistance of $\mathsf{Root}_h(\cdot)$, it then must also be the case that $\tilde{D}'$ is a replay of a previous state in memory in the second split-state. Note that if we assume the leaves of the Merkle tree T are padded with $1.5n \log |\hat{\Sigma}|$ number of random bits (assuming $1.5n \log |\hat{\Sigma}| \geq \ell$ and padded with ℓ bits otherwise), then even given leakage of at most $n \log |\hat{\Sigma}|$ bits (at least a 1/5-fraction of the second split-state), a replay of $\tilde{D}'$ occurs with negligible probability, even for information-theoretic adversaries, since each Merkle Tree has at least $1.5n \log |\hat{\Sigma}|$ number of bits of min-entropy, whereas the attacker can only leak $n \log |\hat{\Sigma}|$ number of bits. Thus, the probability of hitting a previous state in memory is at most $2^{-0.5n \log |\hat{\Sigma}|} \in 2^{-\Omega(\lambda)}$, which is negligible. Therefore, $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ successfully forges with non-negligible probability.

Event EV_2: At some point during the simulator's execution, we have that all of the following hold:

- $\mathcal{I}^{(j+1)} \neq [n]$.
- $R' = R^{(j+1)}$.
- For some $i \in \mathcal{I}^{(j+1)}$, we have that for $\tilde{D}'_i$ and corresponding path p'_i, $R' = \mathsf{Root}_h(p'_i)$.

Claim 7.5. *EV_2 occurs with negligible probability in H_1.*

Proof. The claim is immediate due to collision-resistance of h.

Hybrid H_2: This is exactly the experiment $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D}$.

Claim 7.6. *$H_1 \overset{s}{\approx} H_2$.*

Proof. The claim follows via the observation that, conditioned on EV_1 and EV_2 not occurring, H_1 is identical to H_2. Since EV_1 and EV_2 occur with negligible probability in H_1, we have that H_1, H_2 have at most negligible statistical distance.

Putting everything together, we have that $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},M} \approx \mathbf{Ideal}_{\mathcal{S},\mathcal{U},M}$.

7.2 Full One-Time Tamper Leakage Resilient (OT-TLR)

Definition 15 (NIZK Proof System). *Let R be an efficiently computable binary relation. For pairs $(x, w) \in R$ we call x the statement and w the witness. Let L be the language consisting of statements in R. A proof system for a relation R consists of a crs generation algorithm* (CRSGEN)*, prover* (P) *and verifier* (V)*, which satisfy completeness and soundness properties as follows.*

Definition 16 (Completeness). *For all $x \in L$ and all the witnesses w*

$$\Pr\left[\mathsf{V}(crs, x, \pi \leftarrow \mathsf{P}(crs, x, w)) = 1\right] \geq 1 - negl(\lambda)$$

Definition 17 (Computational Zero-Knowledge). *We call* $(\mathsf{CRSGEN}, \mathsf{P}, \mathsf{V})$ *an NIZK proof for relation R if there exists a polynomial time simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all non-uniform polynomial time adversaries $\mathcal{A}$ we have*

$$\Pr\left[crs \leftarrow \mathsf{CRSGEN}(1^\lambda) : \mathcal{A}^{\mathsf{P}(crs,\cdot,\cdot)}(crs) = 1\right] \overset{c}{\approx} \Pr\left[(crs, \tau) \leftarrow \mathcal{S}_1(1^\lambda) : \mathcal{A}^{\mathcal{S}(crs,\tau,\cdot,\cdot)}(crs) = 1\right]$$

where $\mathcal{S}(crs, \tau, x, w) = \mathcal{S}_2(crs, \tau, x)$ for $(x, w) \in R$ and both oracles output failure if $(x, w) \notin R$.

Definition 18 (Simulation-Sound Extractability). *[44] Consider an NIZK proof of knowledge* $(\mathsf{CRSGEN}, \mathsf{P}, \mathsf{V}, \mathcal{S}_1, \mathcal{S}_2, E_1, E_2)$*. Let* $\mathcal{S}E_1$ *be an algorithm that outputs* (crs, τ, ξ) *such that it is identical to* $\mathcal{S}_1$ *when restricted to the first two parts* (crs, τ)*. We say the NIZK proof is simulation sound if for all non-uniform polynomial time adversaries we have*

$$\begin{aligned}\Pr[(crs,\tau,\xi) \leftarrow & \mathcal{S}E_1(1^\lambda); (x,\pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(crs,\tau,\cdot)}(crs,\xi); w \leftarrow E_2(crs,\xi,x,\pi) : \\ & (x,\pi) \notin Q \text{ and } (x,w) \notin R \text{ and } \mathsf{V}(crs,x,\pi) = 1] \approx 0\end{aligned} \tag{1}$$

where Q *is the list of simulation queries and responses* (x_i, π_i)*.*

Definition 19 (Probabilistically Checkable Proofs). *[12] For functions* $r, q : \mathbb{N} \to \mathbb{N}$ *we say that a probabilistic oracle machine* V *is a* (r, q)*-PCP verifier if, on input a binary string* x *of length* n *and given oracle access to a binary string* π*,* V *runs in time* $2^{O(r(n))}$*, tosses* $r(n)$ *coins, makes* $q(n)$ *queries to* π*, and outputs either 1 (accept) or 0 (reject). A language* L *belongs in the class* $PCP_s[r(n), q(n)]$ *if there exists a* (r, q)*-PCP verifier* V_L *such that the following holds:*

1. *Completeness: If* $x \in L$ *then there exists* π *such that* $\Pr_R[V_L^\pi(x; R) = 1] = 1$*.*
2. *Soundness: If* $x \notin L$ *then for every* π *it holds that* $\Pr_R[V_L^\pi(x; R) = 1] < 1/2$*.*

Theorem 7 (PCP Theorem). *NP = PCP[O(log n), O(1)].*

To achieve negligible soundness, we can run the verifier polylogarithmic number of times in parallel, which results in polylogarithmic number of verifier queries to the proof, π. In [12], they give constructions of PCPs with the above properties, which also allow for knowledge extraction. I.e., assuming $\Pr_R[V_L^\pi(x; R) = 1] \geq 1/2$, there is an efficient extractor which, given π, can extract a witness w for the statement $x \in L$.

We next present our construction:

Construction $\Pi = (\mathsf{ENC}, \mathsf{DEC}, \mathsf{UPDATE})$. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a CCA-secure SS-BRM-PKE scheme, $\mathcal{V} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme in the BRM and H is a family of collision resistance hash functions and Π_{NIZK}, Π_{PCP} which are NIZK and PCP proof systems, respectively. Then we consider the following coding scheme:

- In the pre-processing stage, $crs \leftarrow \mathsf{CRSGEN}(1^\lambda)$ and $h \leftarrow H$ are sampled and $CRS := (crs, h)$ is published. Note that the size of CRS depends on security parameter, but not on the size of the database nor on the leakage parameter ℓ. Note that CRS is implicit input to all algorithms.

- $\mathsf{ENC}(D)$: On input database $D = (D_1, D_2, \ldots, D_n) \in \Sigma^n$:
  - Choose $(\mathsf{pk}, \mathsf{sk}_\varepsilon^1, \mathsf{sk}_\varepsilon^2) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\lambda, 1^\ell)$, where $\mathsf{sk}_\varepsilon^1 \in \hat{\Sigma}^{n'_1}, \mathsf{sk}_\varepsilon^2 \in \hat{\Sigma}^{n'_2}$, and define $\mathsf{sk}_\varepsilon := (\mathsf{sk}_\varepsilon^1 || \mathsf{sk}_\varepsilon^2)$, $(\mathsf{vk}, \mathsf{sk}_\sigma) \leftarrow \mathcal{V}.\mathsf{Gen}(1^\lambda)$.
  - Set $\tilde{D}_0 := 0$.
  - Compute $\tilde{D}_i \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(D_i)$ for $i \in [n]$. Let $\tilde{D} := \tilde{D}_0, \ldots, \tilde{D}_n$. Set[5] $T_{\tilde{D}} := \mathsf{Tree}_h(\tilde{D})$, $\sigma := \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R_{\tilde{D}})$, where $R_{\tilde{D}} := \mathsf{Root}(T_{\tilde{D}})$.
  - Construct the hash tree for secret keys $\mathsf{sk}_\varepsilon^1, \mathsf{sk}_\sigma, T_{\mathsf{sk}^1} = \mathsf{Tree}_h(\mathsf{sk}_\varepsilon^1, \mathsf{sk}_\sigma)$ and $R_{\mathsf{sk}^1} := \mathsf{Root}(T_{\mathsf{sk}^1})$. Repeat the same procedure for secret key $\mathsf{sk}_\varepsilon^2$ and compute $T_{\mathsf{sk}^2} = \mathsf{Tree}_h(\mathsf{sk}_\varepsilon^2)$ and $R_{\mathsf{sk}^2} := \mathsf{Root}(T_{\mathsf{sk}^2})$.
  - For the statement $x_{NIZK} = (\mathsf{pk}, \mathsf{vk})||$ "I know pre-images of hashes $R_{\mathsf{sk}^1}, R_{\mathsf{sk}^2}$)" and witness $w = (\mathsf{sk}_\varepsilon || \mathsf{sk}_\sigma)$ construct the proof

    $$\pi_{NIZK} \leftarrow \Pi_{NIZK}.\mathsf{P}(crs, x_{NIZK}, w)$$

  - For the statement $x_{PCP} =$ "I know an accepting NIZK proof for the statement x_{NIZK}", construct a proof

    $$\pi_{PCP} \leftarrow \Pi_{PCP}.\mathsf{P}(crs, x_{PCP}, \pi_{NIZK})$$

[5] We additionally pad the tree T with dummy leaves consisting of uniform random values to ensure that the relative leakage on the second split state, C_2 is at least $\frac{1}{6}$.

  - Output codeword $C := (C_1, C_2, C_3) \in \hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2} \times \hat{\Sigma}^{n_3}$, where

$$C_1 := (\mathsf{sk}_\varepsilon^1, \mathsf{sk}_\sigma, T_{\mathsf{sk}^1}, R_{\mathsf{sk}^1}) \quad C_2 := (\mathsf{sk}_\varepsilon^2, T_{\mathsf{sk}^2}, R_{\mathsf{sk}^2})$$

$$C_3 := (\mathsf{pk}, \mathsf{vk}, \tilde{D}, T_{\tilde{D}}, R_{\tilde{D}}, \sigma, \pi_{PCP})$$

- $\mathsf{DEC}^C(i)$: On input $i \in [n]$:
  - Parse $C := (\mathsf{sk}_\varepsilon^1, \mathsf{sk}_\sigma, T_{\mathsf{sk}^1}, R_{\mathsf{sk}^1}, \mathsf{sk}_\varepsilon^2, T_{\mathsf{sk}^2}, R_{\mathsf{sk}^2}, \mathsf{pk}, \mathsf{vk}, \tilde{D}, T_{\tilde{D}}, R_{\tilde{D}}, \sigma, \pi_{PCP})$.
  - Check whether $\tilde{D}_0 := 0$. If not, output $\perp$ and terminate.
  - Read path p_i in $T_{\tilde{D}}$, corresponding to leaf i and use p_i to recompute $\hat{R} = \mathsf{Root}_h(p_i)$.
  - Check that $\hat{R} := R_{\tilde{D}}$. If not, output $\perp$ and terminate.
  - Check that $\mathcal{V}.\mathsf{Verify}(\mathsf{vk}, R_{\tilde{D}}, \sigma) = 1$. If not, output $\perp$ and terminate.
  - Run $\Pi_{PCP}.\mathsf{V}(crs, x_{PCP}, \pi_{PCP})$ if outputs 0, output $\perp$ and terminate.
  - For each accessed location of $\mathsf{sk}_\varepsilon^1$ and $\mathsf{sk}_\varepsilon^2$, read the paths in T_{sk^1} and T_{sk^2}, respectively. Compute $\hat{R}_{\mathsf{sk}^1} = \mathsf{Root}(T_{\mathsf{sk}^1})$, $\hat{R}_{\mathsf{sk}^2} = \mathsf{Root}(T_{\mathsf{sk}^2})$ and verify that $\hat{R}_{\mathsf{sk}^1} = R_{\mathsf{sk}^1}$ and $\hat{R}_{\mathsf{sk}^2} = R_{\mathsf{sk}^2}$ for each of them. If any of the verification failed, output $\perp$ and terminate.
  - Output $D_i := \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}_\varepsilon^1 || \mathsf{sk}_\varepsilon^2}(\tilde{D}_i)$.
- $\mathsf{UPDATE}^C(i, v)$: On inputs an index $i \in [n]$, and a value $v \in \Sigma$:
  - Run $\mathsf{DEC}^C(i)$. If it outputs $\perp$, set $\tilde{D}_0 := 1$, write back to memory and terminate.
  - Parse $C := (\mathsf{sk}_\varepsilon^1, \mathsf{sk}_\sigma, T_{\mathsf{sk}^1}, R_{\mathsf{sk}^1}, \mathsf{sk}_\varepsilon^2, T_{\mathsf{sk}^2}, R_{\mathsf{sk}^2}, \mathsf{pk}, \mathsf{vk}, \tilde{D}, T_{\tilde{D}}, R_{\tilde{D}}, \sigma, \pi_{PCP})$.
  - Set $\tilde{D}_i' \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(v)$ Let $\tilde{D}' := \tilde{D}_0, \ldots, \tilde{D}_{i-1}, \tilde{D}_i', \tilde{D}_{i+1}, \ldots, \tilde{D}_n$.
  - Read path p_i in $T_{\tilde{D}}$, corresponding to leaf i and use $(p_i, \tilde{D}_i')$ to compute a new path p_i' (that replaces $\tilde{D}_i$ by $\tilde{D}_i'$). Set $R_{\tilde{D}}' = \mathsf{Root}_h(p_i')$. Let $T_{\tilde{D}}'$ denote the updated tree.
  - Compute $\sigma' := \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R_{\tilde{D}}')$.
  - For each accessed location of sk_σ read the paths in T_{sk^1}. Compute $\hat{R}_{\mathsf{sk}^1} = \mathsf{Root}(T_{\mathsf{sk}^1})$ and verify that $\hat{R}_{\mathsf{sk}^1} = R_{\mathsf{sk}^1}$ for each of them. If any of the verification failed, output $\perp$ and terminate.
  - Write back $(\tilde{D}_i', T_{\tilde{D}}', p_i', R_{\tilde{D}}', \sigma')$ yielding updated codeword $C' := (C_1', C_2', C_3')$ where

$$C_1' := (\mathsf{sk}_\varepsilon^1, \mathsf{sk}_\sigma, T_{\mathsf{sk}^1}, R_{\mathsf{sk}^1}) \quad C_2' := (\mathsf{sk}_\varepsilon^2, T_{\mathsf{sk}^2}, R_{\mathsf{sk}^2})$$

$$C_3' := (\mathsf{pk}, \mathsf{vk}, \tilde{D}', T_{\tilde{D}'}', R_{\tilde{D}'}', \sigma', \pi_{PCP}).$$

Locality of the construction. DEC and UPDATE must read the entire CRS, whose size depends only on security parameter λ and not on the size of the data. In addition, using the SS-BRM-PKE scheme from Section 6.3 (along with sub-exponentially hard PRG), and the PCP of [12], DEC and UPDATE must make $\text{polylog}(\lambda)$ number of random accesses to C.

Remark 3. Note that although computing the PCP proof π_{PCP} is expensive, it is only done a single time, during ENC, but remains static during UPDATE.

Theorem 8. *For security parameter $\lambda \in \mathbb{N}$, leakage parameter $\ell := \ell(\lambda)$, alphabet Σ such that $\log|\Sigma| \in \Omega(\lambda)$, and database size $n := n(\lambda)$: Assume $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is a CCA-secure SS-BRM PKE scheme with leakage parameter $2\ell + \lambda$ and relative leakage $\alpha < 1$, $\mathcal{V} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a signature scheme in the BRM with leakage parameter $2\ell + \lambda$ and relative leakage $\alpha < 1$, H is a family of collision resistant hash functions with sub-exponential security, and Π_{NIZK}, Π_{PCP} are NIZK with simulation-sound extractability and PCP proof systems, respectively. Then Π is a one-time tamper and leakage resilient locally decodable and updatable code taking messages in Σ^n to codewords in $\hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2} \times \hat{\Sigma}^{n_3}$, which is secure against tampering class*

$$\bar{\mathcal{F}} \overset{\text{def}}{=} \left\{ \begin{array}{l} f : \hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2} \times \hat{\Sigma}^{n_3} \to \hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2} \times \hat{\Sigma}^{n_3} \text{ and } |f| \leq \text{poly}(\lambda), \text{ such that :} \\ f = (f_1, f_2, f_3),\ f_1 : \hat{\Sigma}^{n_1} \to \hat{\Sigma}^{n_1},\ f_2 : \hat{\Sigma}^{n_2} \to \hat{\Sigma}^{n_2},\ f_3 : \hat{\Sigma}^{n_3} \to \hat{\Sigma}^{n_3}. \end{array} \right\},$$

and is leakage resilient against the class

$$\bar{\mathcal{G}} \stackrel{\text{def}}{=} \left\{ \begin{array}{l} g : \hat{\Sigma}^{n_1} \times \hat{\Sigma}^{n_2} \times \hat{\Sigma}^{n_3} \to \{0,1\}^{\ell} \times \{0,1\}^{\ell} \times \{0,1\}^{\frac{n_3 \cdot \log|\hat{\Sigma}|}{6}} \ \textit{and}\ |g| \leq \mathrm{poly}(\lambda),\ \textit{such that :} \\ g = (g_1, g_2, g_3),\ g_1 : \hat{\Sigma}^{n_1} \to \{0,1\}^{\ell},\ \ g_2 : \hat{\Sigma}^{n_2} \to \{0,1\}^{\ell},\ g_3 : \hat{\Sigma}^{n_3} \to \{0,1\}^{\frac{n_3 \cdot \log|\hat{\Sigma}|}{6}}. \end{array} \right\}.$$

Moreover, Π has relative leakage $\frac{\alpha}{8} - o(1)$.

Proof (of Theorem 8). To prove the theorem, for any efficient adversary $\mathcal{A}$, we must construct a simulator $\mathcal{S}$, such that for any initial database $D \in \Sigma^n$ and any efficient updater $\mathcal{U}$, the experiment of one time attack $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D}$ is indistinguishable from the ideal experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},D}$.

The simulator $\mathcal{S}$ first samples random coins for the updater $\mathcal{U}$, so its output just depends on its input given the random coins. Then $\mathcal{S}$ works as follows:

- Initially $\mathcal{S}$ samples $(\mathsf{pk}, \mathsf{sk}_{\varepsilon}^1, \mathsf{sk}_{\varepsilon}^2) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^{\lambda}, 1^{\ell}), (\mathsf{vk}, \mathsf{sk}_{\sigma}) \leftarrow \mathcal{V}.\mathsf{Gen}(1^{\lambda}), crs \leftarrow \mathsf{CRSGEN}(1^{\lambda})$ and $h \leftarrow H$, sets $\tilde{D}_0 = 0$ and generates n encryptions of 0, i.e., $\tilde{D}_1, \tilde{D}_2, \ldots, \tilde{D}_n$ where $\tilde{D}_i \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(0)$ for $i \in [n]$. Let $\tilde{D}^{(1)} := \tilde{D}_0, \tilde{D}_1, \ldots, \tilde{D}_n$. $\mathcal{S}$ computes $T_{\tilde{D}}^{(1)} := \mathsf{Tree}_h(\tilde{D}^{(1)})$. Let $\sigma^{(1)} = \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_{\sigma}}(R_{\tilde{D}}^{(1)})$, where $R_{\tilde{D}}^{(1)}$ is the root of the tree $T_{\tilde{D}}^{(1)}$. $\mathcal{S}$ computes $T_{\mathsf{sk}^i} := \mathsf{Tree}_h(\mathsf{sk}^i)$ for $i \in \{1,2\}$, R_{sk^i} denotes the root of the tree T_{sk^i}. $\mathcal{S}$ keeps global variables $\mathsf{flag}, \mathsf{Leaked}, \mathsf{Tampered} = 0$.
- At each round j, let $C^{(j)} := (C_1^{(j)}, C_2^{(j)}, C_3^{(j)})$, where

  $$C_1^{(j)} := (\mathsf{sk}_{\varepsilon}^1, \mathsf{sk}_{\sigma}, T_{\mathsf{sk}^1}, R_{\mathsf{sk}^1}) \quad C_2^{(j)} := (\mathsf{sk}_{\varepsilon}^2, T_{\mathsf{sk}^2}, R_{\mathsf{sk}^2})$$

  $$C_3^{(j)} := (\mathsf{pk}, \mathsf{vk}, \tilde{D}^{(j)}, T_{\tilde{D}}^{(j)}, R_{\tilde{D}}^{(j)}, \sigma^{(j)}, \pi_{PCP}),$$

  denote the current simulated codeword stored by $\mathcal{S}$ and let $w^{(j)}$ denote the simulator's output in the previous round. In the first round, $w_i^{(0)} := \mathit{same}$ for all $i \in [n]$. In each round, $\mathcal{S}$ does the following:

  Simulating Update:

  – If $\mathsf{flag} = 0$, $\mathcal{S}$ does the following:

  - Receives an index $i^{(j)} \in [n]$ from the updater. Runs $\mathsf{UPDATE}^{C^{(j)}}(i^{(j)}, 0)$. Let $C^{(j+1)}$ be the resulting codeword after the update.

  – If $\mathsf{flag} = 1$, $\mathcal{S}$ does the following:

  - Computes $(i^{(j)}, v) \leftarrow \mathcal{U}(\boldsymbol{w}^{(j)})$ on his own, and runs $\mathsf{UPDATE}^{C^{(j)}}(i^{(j)}, v)$. Let $C^{(j+1)}$ be the resulting codeword after the update.

  Simulating the Round's Output:

  – $\mathcal{S}$ sets $(\tilde{D}_0, \tilde{D}_1, \ldots, \tilde{D}_n) := \tilde{D}^{(j+1)}$.

  – $\mathcal{S}$ emulates the adversary $\mathcal{A}$ and receives $g_1, g_2, g_3 \in \bar{\mathcal{G}}$ and $f_1, f_2, f_3 \in \bar{\mathcal{F}}$.

  – If Leaked is 0, then $\mathcal{S}$ computes

  $$\ell_1 := g_1(\mathsf{sk}_{\varepsilon}^1, \mathsf{sk}_{\sigma}, T_{\mathsf{sk}^1}, R_{\mathsf{sk}^1}) \quad \ell_2 := g_2(\mathsf{sk}_{\varepsilon}^2, T_{\mathsf{sk}^2}, R_{\mathsf{sk}^2})$$

  $$\ell_3 := g_3(\mathsf{pk}, \mathsf{vk}, \tilde{D}^{(j+1)}, T_{\tilde{D}}^{(j+1)}, R_{\tilde{D}}^{(j+1)}, \sigma^{(j+1)}, \pi_{PCP})$$

  sets $\ell := (\ell_1, \ell_2, \ell_3)$ and sets Leaked to 1.

  – If $\mathsf{Leaked} = 1$ and $\mathsf{Tampered} = 0$, $\mathcal{S}$ computes $C' = (C_1', C_2', C_3')$ where

  $$C_1' := (\mathsf{sk}_{\varepsilon}^{'1}, \mathsf{sk}_{\sigma}', T_{\mathsf{sk}^1}', R_{\mathsf{sk}^1}') := f_1(C_1) \quad C_2' := (\mathsf{sk}_{\varepsilon}^{'2}, T_{\mathsf{sk}^2}', R_{\mathsf{sk}^2}') := f_2(C_2)$$

  $$C_3' := (\mathsf{pk}', \mathsf{vk}', \tilde{D}', T_{\tilde{D}'}', R_{\tilde{D}'}', \sigma', \pi_{PCP}') := f_3(C_3).$$

  and sets $\mathsf{Tampered}$ to 1.

  – If $\mathsf{flag} = 0$, $\mathcal{S}$ does the following:

- $\mathcal{S}$ sets $\mathcal{I}^{(j+1)} = \{u : \forall u \in [n] \; s.t. \; \tilde{D}'_u \neq \tilde{D}_u \vee \mathsf{DEC}^{C'}(u) = \bot\}$, i.e. the indices where $\tilde{D}'$ is not equal to $\tilde{D}$ or where decode evaluates to $\bot$. $\mathcal{S}$ sets $\mathcal{I}^{(j+1)} = [n]$ if $x'_{NIZK} \neq x_{NIZK}$. If $\mathcal{I}^{(j+1)} = [n]$, $\mathcal{S}$ sets $\mathsf{flag} := 1$. If $\mathcal{I}^{(j+1)} \neq [n]$, $\mathcal{S}$ outputs $\{\ell, \boldsymbol{w}^{(j+1)}\}$, where $w^{(j+1)}[i] = \bot$ for $i \in \mathcal{I}^{(j+1)}$ and $w^{(j+1)}[i] = \mathit{same}$ for $i \notin \mathcal{I}^{(j+1)}$.

– If $\mathsf{flag} = 1$, $\mathcal{S}$ simulates the real experiment faithfully: For $i \in [n]$, $\mathcal{S}$ sets $\boldsymbol{w}^{(j+1)}[i] := \mathsf{DEC}^{(C')}(i)$, i.e. running the real decoding algorithm. Then $\mathcal{S}$ outputs $\{\ell, \boldsymbol{w}^{(j+1)}\}$.

To show $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D} \approx \mathbf{Ideal}_{\mathcal{S},\mathcal{U},D}$, we consider several hybrids.

Hybrid H_0: This is exactly the experiment $\mathbf{Ideal}_{\mathcal{S},\mathcal{U},D}$.

Hybrid H_1: Change π_{PCP} to a simulated proof.

Claim 7.7. $H_0 \overset{c}{\approx} H_1$.

Proof. This claim is due to ZK property of the underlying NIZK.

Event EV_3: $x_{NIZK} \neq x'_{NIZK}$, the verifier accepts, but the extractor fails to extract the witness from π_{PCP}.

Claim 7.8. EV_3 occurs with negligible probability in hybrid H_1.

Proof. The claim is due to the simulation-sound extractability property of the proof system.

Hybrid H_2: We use the knowledge extractor of the PCP and NIZK to extract $\mathsf{sk}'_\varepsilon, \mathsf{sk}'_\sigma$. Everything in the decoding algorithm remains the same up to the final bullet in which the decryption is done by using sk'_ε, instead of using the contents of memory. Everything in the update algorithm also remains the same up to second to last bullet, where signing will now be done with sk'_σ, instead of using the contents of memory.

Claim 7.9. $H_1 \overset{c}{\approx} H_2$.

Proof. This claim is due to collision resistance of h.

Hybrid H_3: The simulator does not encrypt all 0's (i.e. $\mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}}(0)$); instead, it encrypts the real messages.

Claim 7.10. $H_2 \overset{c}{\approx} H_3$.

Proof. Assume there exists an efficient adversary $\mathcal{A}$ distinguishing hybrids H_2 and H_3 with non-negligible advantage. We construct an efficient adversary $\mathcal{A}'$ breaking the $\mathsf{SS\text{-}BRM\text{-}PKE\text{-}CCA}$ security of the encryption scheme $\mathcal{E}$. $\mathcal{A}'$ participates externally in the security game for the SS-BRM PKE scheme (See Section 6 for definition) while internally instantiating $\mathcal{A}$. We next describe $\mathcal{A}'$:

- $\mathcal{A}'$ receives pk from **Key Generation** of its external challenger.
- $\mathcal{A}'$ samples $(\mathsf{vk}, \mathsf{sk}_\sigma) \leftarrow \mathcal{V}.\mathsf{Gen}(1^\lambda)$, $h \leftarrow H$, $(crs', \tau, \xi) \leftarrow SE_1(1^\lambda)$. $\mathcal{A}'$ sets $CRS := (crs', h)$.
- Let $D_1, \ldots, D_n$ denote the initial contents of the database. $\mathcal{A}'$ runs the updater (with fixed coins, as described above) to obtain all the updates $D'_1, \ldots, D'_p$ in advance (where $p := p(\lambda)$ for polynomial $p(\cdot)$ denotes the runtime of the Updater). $\mathcal{A}'$ submits vectors of messages $\boldsymbol{D}_0, \boldsymbol{D}_1$, of dimension $n + p$, as **Message Commitment**. Where $\boldsymbol{D}_0$ is a vector of all 0's and $\boldsymbol{D}_1$ corresponds to the messages as described above.
- $\mathcal{A}'$ instantiates $\mathcal{A}$ on input CRS and waits to receive leakage query (g_1, g_2, g_3) from $\mathcal{A}$.
- Upon receiving leakage query (g_1, g_2, g_3), $\mathcal{A}'$ submits the following **Pre-challenge** split-state leakage query to its challenger:

$$G_1(\mathsf{sk}^1_\varepsilon) := (\mathsf{Root}(\mathsf{Tree}_h(\mathsf{sk}^1_\varepsilon, \mathsf{sk}_\sigma)), g_1(\mathsf{sk}^1_\varepsilon, \mathsf{sk}_\sigma, T_{\mathsf{sk}^1}, R_{\mathsf{sk}^1}))$$
$$G_2(\mathsf{sk}^2_\varepsilon) := (\mathsf{Root}(\mathsf{Tree}_h(\mathsf{sk}^2_\varepsilon), g_2(\mathsf{sk}^2_\varepsilon, T_{\mathsf{sk}^2}, R_{\mathsf{sk}^2})),$$

where $R_{\mathsf{sk}^1} := \mathsf{Root}(\mathsf{Tree}_h(\mathsf{sk}^1_\varepsilon, \mathsf{sk}_\sigma))$ and $R_{\mathsf{sk}^2} := \mathsf{Root}(\mathsf{Tree}_h(\mathsf{sk}^2_\varepsilon))$.

- $\mathcal{A}'$ receives in return the output of its leakage queries $(R_{\mathsf{sk}^1}||\ell_1, R_{\mathsf{sk}^2}||\ell_2)$ as well as challenge ciphertexts $\tilde{D}_i$, $i \in [n]$ and $\tilde{D}'_j$, $j \in [p]$. Let $\tilde{D}^{(1)} := (\tilde{D}_1, \ldots, \tilde{D}_n, \tilde{D}'_1, \ldots, \tilde{D}'_p)$. $\mathcal{A}'$ computes $T^{(1)}_{\tilde{D}} := \mathsf{Tree}_h(\tilde{D}^{(1)})$ and computes $\sigma^{(1)} = \mathcal{V}.\mathsf{Sign}_{\mathsf{sk}_\sigma}(R^{(1)}_{\tilde{D}})$, where $R^{(1)}_{\tilde{D}} := \mathsf{Root}(\mathsf{Tree}_h(\tilde{D}^{(1)}))$. $\mathcal{A}'$ keeps global variables $\mathsf{flag}, \mathsf{Leaked}, \mathsf{Tampered} = \mathsf{0}$.
- $\mathcal{A}'$ uses the simulated proof $\pi'_{NIZK} \leftarrow \mathcal{S}_2(crs', \tau, x_{NIZK} = (R_{\mathsf{sk}^1}, R_{\mathsf{sk}^2})$ to construct the simulated PCP proof π'_{PCP}. Note that $\mathcal{A}'$ now knows the entire contents of the third partition of the codeword, $C_3 := (\mathsf{pk}, \mathsf{vk}, \tilde{D}^{(1)}, T^{(1)}_{\tilde{D}}, R^{(1)}_{\tilde{D}'}, \sigma^{(1)}, \pi'_{PCP}))$. Also note that we assume the proof π'_{PCP} contains the statement x'_{PCP} (and thus also x'_{NIZK}) to be proven, which includes the hash values $R_{\mathsf{sk}^1}, R_{\mathsf{sk}^2}$.
- $\mathcal{A}'$ rewinds $\mathcal{A}$ back to the beginning and instantiates $\mathcal{A}$.
- At each round j, let
  $$C^{(j)}_3 := (\mathsf{pk}, \mathsf{vk}, \tilde{D}^{(j)}, T^{(j)}_{\tilde{D}}, R^{(j)}_{\tilde{D}}, \sigma^{(j)}, \pi'_{PCP})$$
  denote the third partition of the current simulated codeword stored by $\mathcal{A}'$. We maintain the invariant that $\mathcal{A}'$ knows the entire contents of $C^{(j)}_3$, for $j \in [p]$. Let $w^{(j)}$ denote the simulator's output in the previous round. In the first round, $w^{(0)}_i := \mathsf{same}$ for all $i \in [n]$. In each round, $\mathcal{A}'$ does the following:

  Simulating Update:
  - If $\mathsf{flag} = \mathsf{0}$, $\mathcal{A}'$ does the following: Computes the next index $i^{(j)} \in [n]$ generated by the updater. Runs $\overline{\mathsf{UPDATE}}^{C^{(j)}}(i^{(j)}, \perp, \tilde{D}'_j)$. Let $C^{(j+1)}$ be the resulting codeword after the update.
  - If $\mathsf{flag} = \mathsf{1}$, $\mathcal{A}'$ does the following: Computes $(i^{(j)}, v) \leftarrow \mathcal{U}(\boldsymbol{w}^{(j)})$ on his own, and runs $\overline{\mathsf{UPDATE}}^{C^{(j)}}(i^{(j)}, v, \perp)$. Let $C^{(j+1)}$ be the resulting codeword after the update.

  Simulating the Round's Output:
  - $\mathcal{A}'$ sets $(\tilde{D}_0, \tilde{D}_1, \ldots, \tilde{D}_n) := \tilde{D}^{(j+1)}$, $T_{\tilde{D}} := T^{(j+1)}_{\tilde{D}}$, and $R_{\tilde{D}} := R^{(j+1)}_{\tilde{D}}$ and $\sigma = \sigma^{(j+1)}$.
  - $\mathcal{A}'$ emulates the adversary $\mathcal{A}$ and receives $g_1, g_2, g_3 \in \bar{\mathcal{G}}$, $f_1, f_2, f_3 \in \bar{\mathcal{F}}$.
  - If Leaked is 0, then $\mathcal{A}'$ computes $\ell_3 := g_3(C^{(j+1)}_3)$ (recall ℓ_1, ℓ_2 were received previously), returns $\ell := (\ell_1, \ell_2, \ell_3)$ to $\mathcal{A}$ and sets Leaked to 1.
  - If Leaked is 1 and $\mathsf{Tampered}$ is 0, then $\mathcal{A}'$ computes $C'_3 := f_3(C^{(j+1)}_3)$, and sets $\mathsf{Tampered}$ to 1. $\mathcal{A}'$ submits the following **Post-challenge** split-state leakage query to its challenger: $F_1(\mathsf{sk}^1_\varepsilon) := f'_1(\mathsf{sk}^1_\varepsilon, \mathsf{sk}_\sigma, R^{(j+1)}_{\mathsf{sk}^1})$ and $F_2(\mathsf{sk}^2_\varepsilon) := f'_2(\mathsf{sk}^2_\varepsilon, R^{(j+1)}_{\mathsf{sk}^2})$, where f'_1 computes $C'_1 := f_1(C^{(j+1)}_1)$ and then outputs a vector $\boldsymbol{\eta_1} \in \{0,1\}^{n'_1}$ such that for all $i \in [n'_1]$, $\boldsymbol{\eta_1}[i] = 1$ if the path p_i in the Merkle tree in C'_1 is consistent with the root contained in π'_{PCP} and 0 otherwise. Similarly, f'_2 computes $C'_2 := f_2(C^{(j+1)}_2)$ and then outputs a vector $\boldsymbol{\eta_2} \in \{0,1\}^{n'_2}$ such that for all $i \in [n'_2]$, $\boldsymbol{\eta_2}[i] = 1$ if the path p_i in in the Merkle tree in C'_2 is consistent with the root contained in π'_{PCP} and 0 otherwise.
  - $\mathcal{A}'$ additionally receives the **Decryption Access Patterns** for the challenge ciphertexts, i.e. , $(S^1_i, S^2_i) \xleftarrow{Access} \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}^1_\varepsilon||\mathsf{sk}^2_\varepsilon}(\tilde{D}_i)$, $i \in [n]$ and
    $(S^1_j, S^2_j) \xleftarrow{Access} \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}^1_\varepsilon||\mathsf{sk}^2_\varepsilon}(\tilde{D}'_j)$, $j \in [p]$.
  - If $\mathsf{flag} = \mathsf{0}$, $\mathcal{A}'$ does the following:
    - $\mathcal{A}'$ sets $\mathcal{I}^{(j+1)} = \{u : \forall u \in [n]\ s.t.\ \tilde{D}'_u \neq \tilde{D}_u \vee \overline{\mathsf{DEC}}^{C'}(u) = \perp\}$, i.e. the indices where $\tilde{D}'$ is not equal to $\tilde{D}$ or where decode evaluates to $\perp$. $\mathcal{A}'$ checks π'_{PCP} and sets $\mathcal{I}^{(j+1)} = [n]$ if $x'_{NIZK} \neq x_{NIZK}$. If $\mathcal{I}^{(j+1)} = [n]$, $\mathcal{A}'$ sets $\mathsf{flag} := \mathsf{1}$. If $\mathcal{I}^{(j+1)} \neq [n]$, $\mathcal{S}$ outputs $\{\ell, \boldsymbol{w}^{(j+1)}\}$, where $w^{(j+1)}[i] = \perp$ for $i \in \mathcal{I}^{(j+1)}$ and $w^{(j)}[i] = \mathit{same}$ for $i \notin \mathcal{I}^{(j+1)}$.
  - If $\mathsf{flag} = \mathsf{1}$, $\mathcal{A}'$ sets $\boldsymbol{w}^{(j+1)}[i] := \overline{\mathsf{DEC}}^{(C')}(i)$, for $i \in [n]$, and outputs $\{\ell, \boldsymbol{w}^{(j+1)}\}$.
- Once p rounds have completed, $\mathcal{A}'$ outputs whatever $\mathcal{A}$ does and terminates.

$\overline{\mathsf{DEC}}$, $\overline{\mathsf{UPDATE}}$ are defined as follows.

- $\overline{\mathsf{DEC}}^{C}(i)$: On input $i \in [n]$ in round $j \in [p]$:
  - Parse $C_3 := (\mathsf{pk}', \mathsf{vk}', \tilde{D}, T_{\tilde{D}}, R_{\tilde{D}}, \sigma, \pi'_{PCP})$.
  - Check whether $\tilde{D}_0 := 0$. If not, output $\perp$ and terminate.

  - Read path p_i in $T_{\tilde{D}}$, corresponding to leaf i and use p_i to recompute $\hat{R} = \mathsf{Root}_h(p_i)$.
  - Check that $\hat{R} := R_{\tilde{D}}$. If not, output $\perp$ and terminate.
  - Check that $\mathcal{V}.\mathsf{Verify}(\mathsf{vk}', R_{\tilde{D}}, \sigma) = 1$. If not, output $\perp$ and terminate.
  - Run $V_{PCP}(crs', \pi'_{PCP})$ if outputs 0, output $\perp$ and terminate.
  - Let $(\mathcal{S}_j^1, \mathcal{S}_j^2)$ be decryption access patterns; $\forall s^1 \in S_j^1$ and $\forall s^2 \in S_j^2$, if $\boldsymbol{\eta_1}[s^1] = 1$ and $\boldsymbol{\eta_2}[s^2] = 1$ then continue. Else, output $\perp$ and terminate.
  - If $x'_{NIZK} \neq x_{NIZK}$ output $D_i := \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}'_\varepsilon}(\tilde{D}_i)$, where $\mathsf{sk}'_\varepsilon \leftarrow \mathit{\Pi}_{NIZK}.E_2(crs', \xi, \pi_{NIZK})$ is the secret key extracted from the proof π'_{PCP}. E_2 is the witness extractor similar to definition 7.2. Else, compute $D_i := \mathcal{E}.\mathsf{Decrypt}^{\mathsf{sk}_\varepsilon}(\tilde{D}_i)$, by querying the decryption oracle for the CCA secure SS-BRM-PKE with the original secret key.

- $\overline{\mathsf{UPDATE}}^C(i, v, \tilde{D}'_j)$: On inputs an index $i \in [n]$, a value $v \in \mathit{\Sigma}$ and a ciphertext $\tilde{D}'_j \in \widehat{\mathit{\Sigma}}$ in round $j \in [p]$:
  - Run $\overline{\mathsf{DEC}}^C(i)$. If it outputs $\perp$, set $\tilde{D}_0 := 1$, write back to memory and terminate.
  - Parse $C_3 := (\mathsf{pk}', \mathsf{vk}', \tilde{D}, T_{\tilde{D}}, R_{\tilde{D}}, \sigma, \pi'_{PCP})$.
  - If $v = \perp$, Let $\tilde{D}' := \tilde{D}_0, \ldots, \tilde{D}_{i-1}, \tilde{D}'_j, \tilde{D}_{i+1}, \ldots, \tilde{D}_n$. Read path p_i in $T_{\tilde{D}}$, corresponding to leaf i and use $(p_i, \tilde{D}'_j)$ to compute a new path p'_i (that replaces $\tilde{D}_i$ by $\tilde{D}'_j$). Set $R_{\tilde{D}'} = \mathsf{Root}_h(p'_i)$. Let $T_{\tilde{D}'}$ denote the updated tree.
  - If $v \neq \perp$, set $\tilde{D}''_i \leftarrow \mathcal{E}.\mathsf{Encrypt}^{\mathsf{pk}'}(v)$. Otherwise, set $\tilde{D}''_i := \tilde{D}'_j$. Let $\tilde{D}' := \tilde{D}_0, \ldots, \tilde{D}_{i-1}, \tilde{D}''_i, \tilde{D}_{i+1}, \ldots, \tilde{D}_n$. Read path p_i in $T_{\tilde{D}}$, corresponding to leaf i and use $(p_i, \tilde{D}''_i))$ to compute a new path p'_i (that replaces $\tilde{D}_i$ by $\tilde{D}''_i$). Set $R_{\tilde{D}'} = \mathsf{Root}_h(p'_i)$. Let $T_{\tilde{D}'}$ denote the updated tree.
  - Let $S_j^\sigma \xleftarrow{Access} \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R_{\tilde{D}'})$; $\forall s \in S_j^\sigma$, if $\boldsymbol{\eta_1}[s] = 1$ then continue. Else, output $\perp$ and terminate.
  - If $x'_{NIZK} \neq x_{NIZK}$ Compute $\sigma' := \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}'_\sigma}(R'_{\tilde{D}})$, where $\mathsf{sk}'_\sigma \leftarrow E_2(crs, \xi, \pi_{NIZK})$ is the secret key extracted from the proof π'_{PCP}. E_2 is the witness extractor similar to definition 7.2. Otherwise, compute $\sigma' := \mathcal{V}.\mathsf{Sign}^{\mathsf{sk}_\sigma}(R'_{\tilde{D}})$, where where sk_σ is the original secret key.
  - Write back $(\tilde{D}'_i, p'_i, R_{\tilde{D}'}, \sigma')$ yielding updated codeword

$$C'_3 := (\mathsf{pk}', \mathsf{vk}', h, \tilde{D}', T_{\tilde{D}'}, R_{\tilde{D}'}, \sigma', \pi'_{PCP}).$$

If $\tilde{D}_i$, $i \in [n]$ and $\tilde{D}'_j$, $j \in [p]$ are encryptions of all 0's then the view of $\mathcal{A}$ is identical to its view in Hybrid H_2. Alternatively, if $\tilde{D}_i$, $i \in [n]$ and $\tilde{D}'_j$, $j \in [p]$ are encryptions of the honest data values, then the view of $\mathcal{A}$ is identical to its view in Hybrid H_3. Thus, if $\mathcal{A}$ distinguishes with non-negligible advantage, then $\mathcal{A}'$ distinguishes encryptions of all 0's from encryptions of correct data with non-negligible advantage, breaking CCA security of the encryption scheme and resulting in contradiction.

Hybrid H_4: Go back to using $\mathsf{sk}_\varepsilon^1$ and $\mathsf{sk}_\varepsilon^2$ for decryption in case that pk, vk changed and $flag = 1$.

Claim 7.11. $H_3 \overset{c}{\approx} H_4$.

Proof. This claim is true due to collision resistance of h.

Hybrid H_5: Go back to using the real crs and real proof π_{PCP}.

Claim 7.12. $H_4 \overset{c}{\approx} H_5$.

Proof. This claim is due to the zero knowledge property of the proof system.

Hybrid H_6: This is exactly the experiment $\mathbf{TamperLeak}_{\mathcal{A},\mathcal{U},D}$.

Claim 7.13. $H_5 \overset{c}{\approx} H_6$.

Proof. The only difference between H_5 and real experiment is the case where[6]

$$\mathsf{flag} = 0, \tilde{\mathsf{D}}'_{\mathsf{u}} \neq \tilde{\mathsf{D}}_{\mathsf{u}} \text{ and } \mathsf{DEC}^{\mathsf{C}'}(\mathsf{u}) \neq \bot,$$

which can only happen if at some point one of the following events occur:

Event EV_1:

- $\mathsf{pk}'||\mathsf{vk}' = \mathsf{pk}||\mathsf{vk}$. (otherwise $\mathsf{flag} = 1$)
- $\mathcal{I}^{(j+1)} \neq [n]$. (otherwise $\mathsf{flag} = 1$)
- $R' \neq R^{(j+1)}$.
- $\mathsf{Verify}(\mathsf{vk}, R', \sigma') = 1$.

Event EV_2:

- $\mathsf{pk}'||\mathsf{vk}' = \mathsf{pk}||\mathsf{vk}$. (otherwise $\mathsf{flag} = 1$)
- $\mathcal{I}^{(j+1)} \neq [n]$. (otherwise $\mathsf{flag} = 1$)
- $R' = R^{(j+1)}$.
- For some $i \in \mathcal{I}^{(j+1)}$, we have that for $\tilde{D}'_i$ and corresponding path p'_i, $R'_{\tilde{D}} = \mathsf{Root}_h(p'_i)$.

Claim 7.14. EV_1 and EV_2 occur with negligible probability in H_5.

Proof. We omit the proof of the above claim since it is nearly identical to the corresponding claims in proof of Theorem 6.

Both events occur with negligible probability, thus showing that Hybrids H_5 and H_6 differ with negligible probability.

This completes the proof of Theorem 8.

References

1. Aggarwal, D., Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Optimal computational split-state non-malleable codes. In: Kushilevitz and Malkin [54], pp. 393–417
2. Aggarwal, D., Dodis, Y., Kazana, T., Obremski, M.: Non-malleable reductions and applications. In: Servedio, R.A., Rubinfeld, R. (eds.) 47th ACM STOC. pp. 459–468. ACM Press (Jun 2015)
3. Aggarwal, D., Dodis, Y., Lovett, S.: Non-malleable codes from additive combinatorics. In: Shmoys, D.B. (ed.) 46th ACM STOC. pp. 774–783. ACM Press (May / Jun 2014)
4. Aggarwal, D., Dziembowski, S., Kazana, T., Obremski, M.: Leakage-resilient non-malleable codes. In: Dodis and Nielsen [30], pp. 398–426
5. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: Explicit non-malleable codes against bit-wise tampering and permutations. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 538–557. Springer, Heidelberg (Aug 2015)
6. Agrawal, S., Gupta, D., Maji, H.K., Pandey, O., Prabhakaran, M.: A rate-optimizing compiler for non-malleable codes against bit-wise tampering and permutations. In: Dodis and Nielsen [30], pp. 375–397
7. Akavia, A., Goldwasser, S., Vaikuntanathan, V.: Simultaneous hardcore bits and cryptography against memory attacks. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 474–495. Springer, Heidelberg (Mar 2009)
8. Alwen, J., Dodis, Y., Naor, M., Segev, G., Walfish, S., Wichs, D.: Public-key encryption in the bounded-retrieval model. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 113–134. Springer, Heidelberg (May 2010)
9. Alwen, J., Dodis, Y., Wichs, D.: Leakage-resilient public-key cryptography in the bounded-retrieval model. In: Halevi [45], pp. 36–54
10. Alwen, J., Dodis, Y., Wichs, D.: Survey: Leakage resilience and the bounded retrieval model. In: Kurosawa, K. (ed.) ICITS 09. LNCS, vol. 5973, pp. 1–18. Springer, Heidelberg (Dec 2010)

[6] $\mathcal{S}$ would output $\bot$ at position u whereas real experiment would output $\mathsf{DEC}^{C'}(u) \neq \bot$

11. Ball, M., Dachman-Soled, D., Kulkarni, M., Malkin, T.: Non-malleable codes for bounded depth, bounded fan-in circuits. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 881–908. Springer, Heidelberg (May 2016)
12. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.: On the concrete efficiency of probabilistically-checkable proofs. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 585–594. ACM Press (Jun 2013)
13. Brakerski, Z., Kalai, Y.T.: A parallel repetition theorem for leakage resilience. In: Cramer [26], pp. 248–265
14. Canetti, R., Dodis, Y., Halevi, S., Kushilevitz, E., Sahai, A.: Exposure-resilient functions and all-or-nothing transforms. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 453–469. Springer, Heidelberg (May 2000)
15. Cash, D., Ding, Y.Z., Dodis, Y., Lee, W., Lipton, R.J., Walfish, S.: Intrusion-resilient key exchange in the bounded retrieval model. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 479–498. Springer, Heidelberg (Feb 2007)
16. Chandran, N., Goyal, V., Mukherjee, P., Pandey, O., Upadhyay, J.: Block-wise non-malleable codes. Cryptology ePrint Archive, Report 2015/129 (2015), `http://eprint.iacr.org/2015/129`
17. Chandran, N., Kanukurthi, B., Ostrovsky, R.: Locally updatable and locally decodable codes. In: Lindell [56], pp. 489–514
18. Chandran, N., Kanukurthi, B., Raghuraman, S.: Information-theoretic local non-malleable codes and their applications. In: Kushilevitz and Malkin [54], pp. 367–392
19. Chattopadhyay, E., Zuckerman, D.: Non-malleable codes against constant split-state tampering. In: 55th FOCS. pp. 306–315. IEEE Computer Society Press (Oct 2014)
20. Chattopadhyay, E., Zuckerman, D.: Explicit two-source extractors and resilient functions. In: Wichs and Mansour [66], pp. 670–683
21. Cheraghchi, M., Guruswami, V.: Capacity of non-malleable codes. In: Naor, M. (ed.) ITCS 2014. pp. 155–168. ACM (Jan 2014)
22. Cheraghchi, M., Guruswami, V.: Non-malleable coding against bit-wise and split-state tampering. In: Lindell [56], pp. 440–464
23. Choi, S.G., Kiayias, A., Malkin, T.: BiTR: Built-in tamper resilience. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 740–758. Springer, Heidelberg (Dec 2011)
24. Coretti, S., Dodis, Y., Tackmann, B., Venturi, D.: Non-malleable encryption: Simpler, shorter, stronger. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A, Part I. LNCS, vol. 9562, pp. 306–335. Springer, Heidelberg (Jan 2016)
25. Coretti, S., Maurer, U., Tackmann, B., Venturi, D.: From single-bit to multi-bit public-key encryption via non-malleable codes. In: Dodis and Nielsen [30], pp. 532–560
26. Cramer, R. (ed.): TCC 2012, LNCS, vol. 7194. Springer, Heidelberg (Mar 2012)
27. Dachman-Soled, D., Liu, F.H., Shi, E., Zhou, H.S.: Locally decodable and updatable non-malleable codes and their applications. In: Dodis and Nielsen [30], pp. 427–450
28. Di Crescenzo, G., Lipton, R.J., Walfish, S.: Perfectly secure password protocols in the bounded retrieval model. In: Halevi and Rabin [46], pp. 225–244
29. Dodis, Y., Kalai, Y.T., Lovett, S.: On cryptography with auxiliary input. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 621–630. ACM Press (May / Jun 2009)
30. Dodis, Y., Nielsen, J.B. (eds.): TCC 2015, Part I, LNCS, vol. 9014. Springer, Heidelberg (Mar 2015)
31. Dodis, Y., Ristenpart, T., Vadhan, S.P.: Randomness condensers for efficiently samplable, seed-dependent sources. In: Cramer [26], pp. 618–635
32. Dodis, Y., Sahai, A., Smith, A.: On perfect and adaptive security in exposure-resilient cryptography. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 301–324. Springer, Heidelberg (May 2001)
33. Dolev, D., Dwork, C., Naor, M.: Nonmalleable cryptography. SIAM Journal on Computing 30(2), 391–437 (2000)
34. Dziembowski, S.: Intrusion-resilience via the bounded-storage model. In: Halevi and Rabin [46], pp. 207–224
35. Dziembowski, S., Kazana, T., Obremski, M.: Non-malleable codes from two-source extractors. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 239–257. Springer, Heidelberg (Aug 2013)
36. Dziembowski, S., Kazana, T., Wichs, D.: Key-evolution schemes resilient to space-bounded leakage. In: Rogaway [64], pp. 335–353
37. Dziembowski, S., Pietrzak, K.: Intrusion-resilient secret sharing. In: 48th FOCS. pp. 227–237. IEEE Computer Society Press (Oct 2007)
38. Dziembowski, S., Pietrzak, K., Wichs, D.: Non-malleable codes. In: Yao, A.C.C. (ed.) ICS 2010. pp. 434–452. Tsinghua University Press (Jan 2010)
39. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: Continuous non-malleable codes. In: Lindell [56], pp. 465–488

40. Faust, S., Mukherjee, P., Nielsen, J.B., Venturi, D.: A tamper and leakage resilient von neumann architecture. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 579–603. Springer, Heidelberg (Mar / Apr 2015)
41. Faust, S., Mukherjee, P., Venturi, D., Wichs, D.: Efficient non-malleable codes and key-derivation for poly-size tampering circuits. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 111–128. Springer, Heidelberg (May 2014)
42. Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In: Naor [59], pp. 258–277
43. Goyal, V., Pandey, O., Richelson, S.: Textbook non-malleable commitments. In: Wichs and Mansour [66], pp. 1128–1141
44. Groth, J.: Simulation-sound NIZK proofs for a practical language and constant size group signatures. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 444–459. Springer, Heidelberg (Dec 2006)
45. Halevi, S. (ed.): CRYPTO 2009, LNCS, vol. 5677. Springer, Heidelberg (Aug 2009)
46. Halevi, S., Rabin, T. (eds.): TCC 2006, LNCS, vol. 3876. Springer, Heidelberg (Mar 2006)
47. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private circuits II: Keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Heidelberg (May / Jun 2006)
48. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (Aug 2003)
49. Jafargholi, Z., Wichs, D.: Tamper detection and continuous non-malleable codes. In: Dodis and Nielsen [30], pp. 451–480
50. Kalai, Y.T., Kanukurthi, B., Sahai, A.: Cryptography with tamperable and leaky memory. In: Rogaway [64], pp. 373–390
51. Katz, J., Trevisan, L.: On the efficiency of local decoding procedures for error-correcting codes. In: 32nd ACM STOC. pp. 80–86. ACM Press (May 2000)
52. Katz, J., Vaikuntanathan, V.: Signature schemes with bounded leakage resilience. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 703–720. Springer, Heidelberg (Dec 2009)
53. Kiayias, A., Liu, F.H., Tselekounis, Y.: Practical non-malleable codes from l-more extractable hash functions. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 16. pp. 1317–1328. ACM Press (Oct 2016)
54. Kushilevitz, E., Malkin, T. (eds.): TCC 2016-A, Part II, LNCS, vol. 9563. Springer, Heidelberg (Jan 2016)
55. Lindell, Y.: A simpler construction of cca2-secure public-key encryption under general assumptions. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 241–254. Springer, Heidelberg (May 2003)
56. Lindell, Y. (ed.): TCC 2014, LNCS, vol. 8349. Springer, Heidelberg (Feb 2014)
57. Liu, F.H., Lysyanskaya, A.: Tamper and leakage resilience in the split-state model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 517–532. Springer, Heidelberg (Aug 2012)
58. Micali, S., Reyzin, L.: Physically observable cryptography (extended abstract). In: Naor [59], pp. 278–296
59. Naor, M. (ed.): TCC 2004, LNCS, vol. 2951. Springer, Heidelberg (Feb 2004)
60. Naor, M., Segev, G.: Public-key cryptosystems resilient to key leakage. In: Halevi [45], pp. 18–35
61. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: 22nd ACM STOC. pp. 427–437. ACM Press (May 1990)
62. Pietrzak, K.: A leakage-resilient mode of operation. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 462–482. Springer, Heidelberg (Apr 2009)
63. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 84–93. ACM Press (May 2005)
64. Rogaway, P. (ed.): CRYPTO 2011, LNCS, vol. 6841. Springer, Heidelberg (Aug 2011)
65. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: 40th FOCS. pp. 543–553. IEEE Computer Society Press (Oct 1999)
66. Wichs, D., Mansour, Y. (eds.): 48th ACM STOC. ACM Press (Jun 2016)