Random Robust Secret Sharing with Perfect Privacy and its Applications

Mohammad Hassan Ameri Jeremiah Blocki

Purdue University
West Lafayette, IN USA

Abstract

Secret Sharing schemes allow a dealer to distribute n shares $s_1, \ldots, s_n$ of a secret s so that any t shares suffice to reconstruct the secret, while any $t-1$ shares reveal no information about s. In fact, schemes such as Shamir Secret Sharing satisfy a stronger guarantee called $(t-1)$-perfect privacy, meaning that for any subset $S \subseteq [n]$ with $|S| \leq t-1$, the joint distribution $(s_i)_{i \in S}$ is uniformly distributed over its domain. This strong guarantee is essential for applications such as fuzzy password-authenticated key exchange (fPAKE) and conditional encryption — a recent cryptographic primitive introduced to enable secure personalized password typo correction. Unfortunately, Shamir secret sharing is not robust: corrupted shares can prevent correct reconstruction or cause reconstruction of an incorrect secret. Existing robust secret sharing schemes address this issue but necessarily sacrifice perfect privacy. We introduce and construct *Random Robust Secret Sharing with Perfect Privacy* (RRSS), a new notion that preserves $(t-1)$-perfect privacy while providing robustness against random share corruptions. In our schemes, the secret is recovered with high probability even if an arbitrary subset of up to $n-t$ shares is independently corrupted at random. We demonstrate the utility of RRSS through two applications. First, we present the first practically efficient fPAKE construction that tolerates Hamming errors. Second, we obtain the first efficient conditional encryption scheme for arbitrary Hamming distances, improving upon prior work that achieved efficiency only for constant distances. We implement both constructions and empirically demonstrate their practicality.

Full Version: This document provides the full version of the article published at CCS 2026 under the same title — see https://doi.org/10.1145/3830454.3846816.

1 Introduction

Secret sharing schemes [Sha79] are foundational cryptographic primitives which allows a dealer disperse a secret among n parties so that any subset of t parties can collaborate to recover the secret, whilst any subset of *at most* $t-1$ parties will fail to recover the secret. More precisely, in a t out of n secret sharing scheme, the dealer will distribute n shares $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n)$ of the secret s among n players such that any subset of t parties can collaborate to reconstruct the secret s, while any combination of at most $t-1$ parties cannot learn anything about the secret in an information theoretic sense. Some secret sharing schemes (including Shamir's [Sha79]) satisfy an even stronger notion called $(t-1)$-perfect privacy which intuitively ensures that for any subset $S \subseteq [n]$ of at most $|S| \leq t-1$ shares, the joint distribution $(\llbracket s \rrbracket_i)_{i \in S}$ is uniform. This condition is sufficient, but not always necessary, to ensure that any subset of $t-1$ parties cannot learn anything about the secret. The stronger $(t-1)$-perfect privacy requirement can be useful as it additionally ensures that, if $n-t+1$ shares are randomly corrupted, then it is impossible to distinguish corrupted shares from correct shares.

In some applications (e.g., secure multi-party computation [BP16] and secure cloud storage [CDD+15], secure message transmission [DDWY90, GGO10, GGOR14], distributed algorithm [CFOR12]) it is useful to ensure

that the Secret Sharing scheme share recovery algorithm is *robust* to a few incorrect shares [RB89, CDV94, TW88]. This ensures that it is possible to identify and remove incorrect shares if some parties shares were corrupted either maliciously or randomly. Intuitively, if we receive n shares and at most $n-t$ of these shares were corrupted then we would like to be able to (1) identify all corrupted shares, and (2) recover the secret using t of the remaining uncorrupted shares. There has been a line of work on designing and improving robust secret sharing schemes (e.g., [CDF01, CDF+08, CFOR12, Che19, BP16, BPRW16, HO18, FY19]). Each of these constructions utilizes an information theoretic MAC which is embedded in individual shares and then used to identify dishonest parties who report an incorrect share. They all use a pairwise MAC between all pairs of parties so that each honest party i can independently (in)validate the share from any other party j. This is a useful property in some MPC applications as it allows each honest party to independently identify parties that behaved dishonestly and recover the correct secret. However, this property also runs directly counter to the stronger guarantee of $(t-1)$-perfect privacy. In particular, a subset $(\llbracket s \rrbracket_i, \llbracket s \rrbracket_j)$ of just $2 \leq t$ valid shares can already be trivially distinguished from a uniformly random string simply by validating the MAC.

There are some applications where the stronger guarantee of perfect privacy is required. As an example we consider the (relaxed) Fuzzy Password-based Authenticated Key Exchange (fPAKE) protocol from [DHP+18, BFH+23]. Intuitively, an (exact) PAKE protocol allows Alice and Bob to securely agree on a session key K as long as they share a (possibly low-entropy) password `pwd` — if the passwords don't match then Alice and Bob should not learn anything about the other party's password. By contrast, a fPAKE allows Alice and Bob to agree on the session key K as long as their passwords are close under some distance metric e.g., even if their passwords don't exactly match they should be able to exchange a key as long as the hamming distance between the two passwords is below a threshold ℓ. Two concrete constructions of fPAKE for Hamming distance are presented in [DHP+18] and [BFH+23]. Intuitively, these protocols work by having one party (Alice) generate n shares $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n)$ of a secret session key K where n is the number of characters in the password. Then for each individual character Alice and Bob run a special PAKE protocol to exchange each secret share $\llbracket s \rrbracket_i$. If the passwords agree on character i then Bob will receive the correct share $\llbracket s \rrbracket_i$ otherwise Bob will receive a randomly corrupted share $\llbracket s' \rrbracket_i$. The key idea is that Bob will obtain enough shares to recover the session key K if and only if the Hamming Distance between the passwords is at most $n-t$. However, the above description is overly simplified and has a problem. If the secret sharing scheme is not robust then Bob will not be able to identify the corrupted shares and recover the secret key K. It is tempting to simply fix the problem by using a robust secret sharing scheme. However, this introduces a major security issue. Suppose for example, that Bob's password only matches Alice's password on the first character so that Bob receives the uncorrupted share $\llbracket s \rrbracket_i$ and randomly corrupted shares $\llbracket s' \rrbracket_j$ for all $j > 1$. Bob can use the MACs embedded in $\llbracket s \rrbracket_1$ to conclude that the first character matches while each the remaining characters are incorrect. Thus, even though the passwords do not match Bob learns a lot of information about Alice's password — over time a malicious Bob would be able to quickly guess each individual character of Alice's password.

Dupont et al. [DHP+18] address this issue by constructing a new secret sharing scheme achieving t_2-robustness and t_1-perfect privacy with $t_1 < t_2 - 1$. Intuitively, t_2-robustness means that if we are given n shares and at most $n - t_2$ of those shares are corrupt then it is possible to identify the corrupted shares and recover the secret. t_1-perfect privacy means that for any subset $S \subseteq [n]$ of $|S| \leq t_1$ shares that the joint distribution $(s_i)_{i \in S}$ is uniformly random. However, the secret sharing scheme of Dupont et al. [DHP+18] has a gap between the two parameters t_2 and t_1 i.e., $t_1 < t_2 - 1$. Bootle et al. [BFH+23] reduced the size of the gap between t_2 and t_1 but there is still a gap. Because of this gap, one cannot truly consider the protocol of Dupont et al. [DHP+18] (or [BFH+23]) to be an fPAKE, since it only achieves a relaxed/weakened security notion. In the (relaxed) fPAKE protocol Bob will only be able to recover the session key K if the Hamming Distance between the two passwords is at most $n - t_2$, so that at least t_2 characters match. However, the security guarantee only prevents Alice/Bob from learning useful information about the other's password if the hamming distance is at least $n - t_1 \geq n - t_2 + 2$. Because user chosen passwords typically have low-entropy [Bon12, BL23], even a one-character increase in the Hamming distance at which privacy

is guaranteed can substantially weaken security. For example, suppose that an fPAKE protocol permits successful key exchange when the two passwords have Hamming distance at most 2. If Alice's password is `123ABC` and the attacker (Bob) submits the common password `123456`, then key exchange should fail as the two passwords have hamming distance 3. An ideal fPAKE security guarantee ensures that the attacker learns nothing about the Alice's password beyond the fact that its hamming distance from `123456` exceeds 2. By contrast, a relaxed fPAKE protocol with a one-character privacy/robustness gap provides no privacy guarantee in this case; and its security definition does not even rule out possibility that the attacker learns Alice's password in its entirety.

Ideally we would like to have a secret sharing scheme that is t_2-robust and t_1-perfectly private with no gap i.e., $t_2 = t_1 + 1$. Unfortunately, this ideal setting with $t_2 = t_1 + 1$ is impossible if one requires strong robustness against adversarial corruptions — see Appendix I.2.

A very recent and concurrent work by Richardson et al. [RRX26] presents a theoretical construction of an fPAKE protocol based on garbled circuits, improving the efficiency of a protocol of Dupont et al. [DHP+18] only a single garbled circuit (instead of two). While the protocol achieves full fPAKE security, the paper does not provide an implementation or empirical evaluation. Moreover, garbled-circuit-based solutions are unlikely to yield practical instantiations in this setting due to their substantial computational and communication overhead, as well as the need to encode passwords at the bit level rather than the character level. Thus, the question of constructing a practically efficient fPAKE protocol achieving full (non-relaxed) security guarantees remains open.

Conditional Encryption [AB24] provides yet another example where the stronger guarantee of $(t-1)$-perfect privacy is critical. Conditional Encryption is a primitive introduced by Ameri and Blocki [AB24] to enable secure personalized password typo correction. Intuitively, a conditional encryption scheme is a public key encryption scheme augmented by the addition of a new conditional encryption algorithm CEnc. The conditional encryption algorithm takes as input the public key pk, a ciphertext c_1 encrypting an unknown message m_1, a control message m_2 and a payload message m_3. The output is a conditional ciphertext c with the guarantee that (1) if the unknown message m_1 is related to the control message m_2 (i.e., $P(m_1, m_2) = 1$) then c will decrypt to the payload message m_3, (2) unknown message m_1 is not related to the control message m_2 (i.e., $P(m_1, m_2) = 0$) then the conditional ciphertext will not leak any information the control message or the payload message — even if the adversary knows the secret decryption key sk. Ameri and Blocki [AB24] constructed a conditional encryption scheme for the Hamming Distance predicate $P_{\mathtt{Ham}\leq\ell}(m_1, m_2)$ which is 1 if and only if the hamming distance between m_1 and m_2 is at most ℓ. Their construction utilized Pallier Public Key Encryption and Shamir Secret Sharing. Intuitively, the conditional encryption algorithm will generate n shares of a fresh symmetric key K and work character by character to generate conditional encryptions of each share. The conditional encryption algorithm additionally encrypts the payload message m_3 using K. As long as the Hamming Distance is at most ℓ the decryption algorithm will recover enough shares to recover K and decrypt m_3. The security argument crucially relies on the observation that Shamir Secret Sharing achieves $(t-1)$-perfect privacy to ensure that an attacker cannot learn whether or not $m_1[i] = m_2[i]$ by distinguishing the i'th share of K from a random value when the Hamming Distance between m_1 and m_2 is large. However, because Shamir Secret Sharing is not robust there is not an efficient way to identify remove the $n-t$ corrupted shares. Ameri and Blocki [AB24] addressed the challenge by simply brute-forcing over all possible $\binom{n}{n-t}$ subsets to find the correct key K. Thus, their construct is only efficient when the Hamming distance $\ell = n - t$ is a small constant, e.g., their primary implementation uses $\ell = 2$. Once again if we had a robust secret sharing scheme with perfect privacy we would be able to greatly improve the construction of Ameri and Blocki [AB24] to obtain an efficient scheme for arbitrary Hamming Distances.

A key observation underlying both applications is that the robustness requirement is fundamentally a correctness concern, whereas perfect privacy is a security concern. In the settings of interest—namely fuzzy PAKE and conditional encryption—security must hold against fully malicious adversaries, but correctness is only required when both parties follow the protocol honestly.

Crucially, even in honest executions, it is entirely expected that some shares will be corrupted. In

fPAKE, this occurs whenever the two parties' passwords differ at a particular position: the corresponding share s_i obtained by the receiving party is effectively corrupted. Similarly, in conditional encryption for Hamming distance, character mismatches induce incorrect shares during decryption. However, in both cases these corrupted shares are not adversarially chosen. In the fPAKE application, even in the presence of malicious behavior, character mismatches induce independent and uniformly random corruptions. In the conditional-encryption application, which considers semi-honest adversaries, character mismatches similarly produce independently and uniformly random shares. This distinction is critical. Existing notions of robust secret sharing are designed to tolerate arbitrary (maliciously crafted) share corruption, which necessitates heavy machinery such as pairwise MACs and inevitably conflicts with $(t-1)$-perfect privacy. Prior work [DHP+18, BFH+23] focused on reducing the gap between t_2 (robustness) and t_1 (perfect privacy). By contrast, we ask whether or not the gap can be eliminated entirely.

"*Can we eliminate the privacy/robustness gap by relaxing the robustness guarantee?*"

More specifically, we ask whether or not it is possible to design a secret sharing scheme with $(t-1)$-perfect privacy while maintaining robustness against up to $n-t$ randomly corrupted shares. We answer this question in the affirmative by introducing the new notion of *Random Robust Secret Sharing* (RRSS), and providing an efficient construction for any value of t and n. We also ask whether or not such a relaxed robustness constraint is sufficient for our applications: fPAKE and Conditional Encryption. Once again we answer this question in the affirmative.

1.1 Our Contributions

The contributions of the paper are outlined as follows.

1.1.1 Introducing Random Robust Secret Sharing

We introduce *Random Robust Secret Sharing* (RRSS) and formalize its security guarantees. An RRSS scheme Π_{RRSS} consists of four efficient algorithms $\Pi_{\mathsf{RRSS}} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$. The setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, t)$ takes as input the security parameter λ along with n and t and generates public parameters pp which can be published. In our construction $\mathsf{Setup}(1^\lambda)$ can actually be deterministic, eliminating the need for trusted step. To share a secret s, the dealer computes a vector of n shares $\vec{s} = (\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$. The $\mathsf{Identify}$ algorithm is used to identify t uncorrupted shares which can then be used to recover the secret.

A bit more formally, we say that Π_{RRSS} is an $(n, t, \epsilon(\lambda))$-RRSS scheme (see Definition 1) if it is secure against random share corruption and satisfies two properties: $(t-1)$-perfect privacy and t-random robustness. $(t-1)$-perfect privacy requires that the joint distribution of any subset of at most $t-1$ shares is uniform and independent of the secret. The t-random robustness property states that for any $S \subseteq [n]$ of size $|S| = n - t$ we have

$$\Pr_{\vec{s}'}\left[\mathsf{SecretRecover}\left(\vec{s}', \mathsf{pp}\right) = s \mid \llbracket s' \rrbracket_i = \llbracket s \rrbracket_i \ \ \forall i \notin S\right] \geq 1 - \epsilon(\lambda)$$

where the randomness is taken over the uniformly random selection of $\llbracket s' \rrbracket_i$ for each $i \in S$. This means that if up to $n-t$ shares are randomly corrupted we will still be able to recover the secret with high probability — we will pick $\epsilon(\lambda)$ negligible. The first step for the $\mathsf{SecretRecover}$ procedure is to use $\mathsf{Identify}$ to find t uncorrupted shares. The $\mathsf{Identify}$ algorithm ensures that for any $S \subseteq [n]$ of size $|S| = n - t$ we have

$$\Pr_{\vec{s}'}[|U| \geq t \wedge U \subseteq [n] \setminus S \mid U \leftarrow \mathsf{Identify}\left(\vec{s}', \mathsf{pp}\right) \\ \wedge \llbracket s' \rrbracket_i = \llbracket s \rrbracket_i \ \ \forall i \notin S] \geq 1 - \epsilon(\lambda)$$

where the randomness is taken over the uniformly random selection of $\llbracket s' \rrbracket_i$ for each $i \in S$. In particular, if $n-t$ shares are randomly corrupted we will still be able to correctly and efficiently identify the t remaining shares that are correct.

As an additional (optional) property we say that the RRSS scheme is secure against malicious dealers if for all strings $[\![s']\!]_1, \ldots, [\![s']\!]_n$ and all $S \subseteq [n]$ of size $|S| > n - t$ we have

$$\Pr_{\vec{s}''}\left[\mathsf{Identify}\left(\vec{s}'', \mathsf{pp}\right) = \perp \mid \vec{s}''\left[i\right] = [\![s']\!]_i \quad \forall i \notin S\right] \geq 1 - \epsilon(\lambda)$$

where the randomness is taken over the uniformly random selection of $\vec{s}''_i$ for each $i \in S$. In particular, this means that if more than $n - t$ shares are randomly corrupted, then $\mathsf{Identify}$ will output $\perp$ — even if the initial shares $[\![s']\!]_1, \ldots, [\![s']\!]_n$ were generated by a malicious dealer. See Definition 1 for a formal description of $(n, t, \epsilon(\lambda))$-RRSS scheme.

1.1.2 Construction of RRSS with formal security proof

We show how to construct a $(n, t, \epsilon(\lambda))$-random robust secret sharing which provides $(t-1)$-perfect privacy and t-robustness against random share corruption. Our construction completely eliminates the gap between the perfect privacy parameter $t_1 = t-1$ and the robustness parameter $t_2 = t_1+1 = t$ from prior constructions of Robust Secret Sharing with Perfect Privacy [DHP+18, BFH+23]. In particular, either t shares remain uncorrupted and we can fully recover the secret or more than $n-t$ shares are randomly corrupted, in which case the remaining shares are uniformly random and leak no information about the secret *or about which shares were corrupted*. As our primary building block, our construction utilizes Shamir Secret Sharing. In the setup phase we pick $3n$ arbitrary non-zero field elements $a_1, \ldots, a_{3n} \neq 0$ which are included in the public parameters pp. The $\mathsf{ShareGen}$ generation algorithm will generate n shares $([\![s]\!]_1, \ldots, [\![s]\!]_n)$ where each individual share $[\![s]\!]_i = ([\![s]\!]_i^0, \ldots, [\![s]\!]_i^{6n})$ consists of a vector of $6n+1$ Shamir Secret Shares. Here, the shares $([\![s]\!]_1^0, \ldots, [\![s]\!]_n^0)$ are generated using a regular (n, t)-shamir secret sharing scheme to generate n shares of the underlying secret s. For $j \geq 1$ the shares $[\![s]\!]_1^{2j-1}, [\![s]\!]_1^{2j}, \ldots, [\![s]\!]_i^{2j-1}, [\![s]\!]_i^{2j}, \ldots, [\![s]\!]_n^{2j-1}, [\![s]\!]_n^{2j}$ are generated using a $(2n, 2t)$-shamir secret sharing to generate $2n$ shares of the public value $a_j \neq 0$ which is fixed a priori. Intuitively, the first part $[\![s]\!]_i^0$ of each share $[\![s]\!]_i$ is used to recover the secret. The remaining parts $([\![s]\!]_i^1, \ldots, [\![s]\!]_i^{6n})$ are used to help efficiently identify and eliminate incorrect shares using Gaussian Elimination. We prove that if $n-t$ shares are randomly corrupted while the remaining t shares are correct then with high probability the solution to our linear system of equations will correctly reveal the location of at least t valid shares. Once we have identified a set S of t valid shares we can use the regular polynomial interpolation to recover the secret using the secrets $\{[\![s]\!]_i^0\}_{i \in S}$. Our construction is also secure against malicious dealers i.e., if more than $n-t$ shares are randomly corrupted then (whp) the identification algorithm will recognize this and output $\perp$ and this holds even if the initial shares were generated by a malicious dealer. See Section 3 for the definition of RRSS, a full description of our construction and security proofs.

1.1.3 Application 1: Efficient fPAKE Construction for Hamming Distance

As the first application of our RRSS, we use it to construct an efficient Fuzzy Password Authenticated Key Exchange (fPAKE) protocol [DHP+18, CHK+05, BFH+23] for Hamming Distance. A PAKE protocol allows two parties who share a (low-entropy) password to securely exchange a cryptographic key — key exchange is successful if and only if the passwords match exactly. In an fPAKE protocol, the key exchange process should be successful, if the passwords are "sufficiently close". If the key-exchange is not successful, then participants should not learn anything about the password of the other party other than that it was not "sufficiently close" to his/her own. We consider two passwords to be "sufficiently close" if Hamming Distance between the two passwords is at most $n-t$. As we noted the fPAKE protocols from [DHP+18, BFH+23] only achieve a relaxed security notion i.e., success is only guaranteed if the Hamming Distance is at most $n-t_2$ while security is only guaranteed if the Hamming Distance is at most $n-t_1 > n-t_2+1$. The gap arises due to the use of Robust Secret Sharing.

Equipped with our new notion of random robust secret sharing we can eliminate the gap entirely. In particular, we observe that robustness to fully adversarial corruptions is unnecessary. The fPAKE protocols from [DHP+18] and [BFH+23] use an underlying iPAKE protocol to exchange a key K_i for each character $i \in [n]$ of the password — the key K_i is then used as a one-time pad to transmit the i'th secret share of the final secret key K. If the passwords do not match at location i then ideal functionality for a UC-secure

iPAKE ensures that the key K_i is randomly (not adversarially) corrupted. Effectively, we can simply take the protocol from [BFH+23] and replace the Robust Secret Sharing scheme with a Random Robust Secret Sharing scheme. In particular, if the Hamming Distance between the two passwords is at most $n-t$ then key exchange will succeed while if the Hamming Distance is greater than $n-t$ then neither party learns anything about the password of the other party (other than that the Hamming Distance is greater than $n-t$). We prove that our construction UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$ for fPAKE. We follow the definition from [BFH+23] though we only consider the special case where there is no privacy/robustness gap i.e., $t_2 = t_1 + 1$.

Fomichev et al. [FHA+21] implemented the (relaxed) fPAKE protocol from [BFH+23] as part of the Fast-ZIP protocol, which leverages physical context (e.g., ambient audio) to enable secure device pairing without user interaction. We modify this implementation by replacing RSS with our RRSS construction, thereby upgrading the protocol to achieve full fPAKE security. To the best of our knowledge, this is the first practical implementation of an fPAKE protocol satisfying the full (non-relaxed) ideal functionality — the fPAKE protocol from [RRX25] achieves full fPAKE security, but has not been implemented or empirically evaluated.

We evaluate the performance of our modified protocol in terms of execution time, communication overhead, and overall delay. For fingerprints of length $n = 36$, the sender (resp. receiver) execution time is 0.2498 (resp. 0.0447) seconds, with overall delay 0.2580 (resp. 0.2943) seconds and communication overhead 34.20318 (resp. 1.8281) KB when parameters are selected to achieve $\lambda = 128$-bit security.

For comparison, the relaxed fPAKE implementation of [FHA+21] achieves sender (resp. receiver) execution time 0.0123 (resp. 0.0485) seconds, overall delay 0.0186 (resp. 0.0605) seconds, and communication overhead 3.3008 (resp. 1.82) KB. While our construction incurs higher overhead due to the larger RRSS share size and linear-algebra-based identification step, the performance remains practical and enables stronger security guarantees.

See Section 4 for a detailed construction, and a brief overview of the UC security definition and proof. See Appendix F.2 for the full proof.

1.1.4 Application 2: Conditional Encryption for Arbitrary Hamming Distance

As the second application of our RRSS, we use it to provide an *efficient* and secure conditional encryption scheme for Hamming Distance with arbitrarily large distance parameters ℓ. In particular, our conditional decryption algorithm runs in time $O(n^\omega)$ i.e., the cost to solve a system of linear equations of size $O(n) \times O(n)$. By contrast, the conditional decryption algorithm of [AB24] runs in time $\Omega\left(\binom{n}{\ell}\right)$ which is exponential if ℓ is large and is only practical for very small hamming distance parameters e.g., $\ell \in \{1, 2, 3, 4\}$. We obtain our more efficient conditional encryption construction by modifying the construction of [AB24] to replace the t out of n Shamir Secret Sharing scheme with a Random Robust Secret Sharing Scheme. Because each RRSS secret share in our constructions consists of $O(n)$ Shamir shares, the length of a conditional ciphertext scales with $O(n^2)$. This is longer than the $O(n)$ term from [AB24], but still manageable. Empirical analysis shows that for typical values of n the final size of a conditional ciphertext size is comparable to [AB24]. This is due to efficient share packing optimization where we show how to pack multiple Shamir Shares into a single Pallier ciphertext.

We implement our conditional encryption construction and evaluate its performance empirically. Our results demonstrate that the scheme remains practically efficient even for larger values of ℓ. For passwords of length at most $n = 32$, the ciphertext size is 48.08 KB and the conditional decryption time is 1.208 seconds. By contrast, the construction of [AB24] achieves ciphertext size 24.08 KB and decryption time 14.4 minutes for Hamming distance threshold $\ell = 16$. Moreover, its running time grows combinatorially in ℓ, becoming impractical as n and ℓ increase, and is competitive only when $\ell \leq 2$.

See Section 5 for an overview of the construction and security proofs. Full technical details are provided in Appendix H.

Implementations and Replicability: To support replicability, we plan to release all source code as an open-source artifact. For the review process, the implementation is temporarily available via an anonymous repository at https://github.com/mhassanameri/RRSS. The repository includes complete implementations of RRSS, our fPAKE protocol, and our Conditional Encryption scheme for arbitrary Hamming distance.

2 Preliminaries

In this section, we review the notations and cryptographic primitives which will be used in the rest of the paper.

Given a set S we use $s \in_R S$ to denote a uniformly random sample from S. Given a randomized algorithm $\mathcal{A}$ we use the notation $y = \mathcal{A}(x; r)$ to denote the deterministic output y when we fix the random coins r and we use $y \leftarrow \mathcal{A}(x)$ to denote the randomized output when the random coins r are sampled uniformly at random.

Let Σ denote an alphabet (e.g., ASCII or unicode). Given a string $w \in \Sigma^*$ we use $\|w\|$ to denote the length of w and for $i \leq \|w\|$ we use $w[i]$ to denote the ith character of w. We let $\mathcal{M}_n = \Sigma^{\leq n}$ denote the set of all strings w with length $\|w\| \leq n$ and Σ^n denotes the set of all strings of length n.

Given an integer $n \geq 1$ we use $[n] = \{1, \ldots, n\}$. Given a vector $\vec{s}$ of length n and a subset $S \subseteq [n]$ we use $\vec{s}_S = (\vec{s}[i])_{i \in S}$ to denote the subvector of length $|S|$ corresponding to the indices in S. Let $L = \langle l_1, \ldots, l_{|L|} \rangle$ be list of $|L|$ elements. We also define the operation $L' = \texttt{Append}(L, l)$ which adds l to the list and we have $L' = \langle l_1, \ldots, l_{|L|}, l \rangle$. We note that l_i can be an element in $\mathbb{Z}_{N^2}, \Sigma^n, \mathcal{M}_n$ or $\mathcal{PWD}$, etc.

2.1 Hamming Distance and Close Passwords

Given two strings $w_1, w_2 \in \Sigma^n$ we use $\texttt{Ham}(w_1, w_2) = |\{i | w[i] \neq w[j]\}|$ to denote the hamming distance between them. Note that if $w_1 = w_2$ then $\texttt{Ham}(w_1, w_2) = 0$. In this paper we use Hamming Distance to determine if two passwords $\texttt{pwd}_1, \texttt{pwd}_2$ are close e.g., we define a predicate $P_t(\texttt{pwd}_1, \texttt{pwd}_2) = 1$ if $\texttt{Ham}(\texttt{pwd}_1, \texttt{pwd}_2) \leq t$ for some fixed threshold $t \leq n$.

It will be convenient to assume that all passwords have the same length. While user passwords are generally short, thy do not all have the exact same length. Thus, we define the 1 to 1 function $\texttt{Pad} : \Sigma^{\leq n-1} \rightarrow \Sigma^n$ and consider $\mathcal{PWD} = \Sigma^n$ to be the set of all possible length $\leq n-1$ passwords after padding. In practice, it would be reasonable to select $n = 30$ as essentially all user passwords are shorter than this (e.g., over 99.9% of leaked RockYou passwords were less than 30 characters.). The symbol "$\|$" will be used for concatenation. Thus, $y = x_1 \| x_2$ is concatenation of x_1 and x_2. Given two strings $\texttt{pwd}_1, \texttt{pwd}_2 \in \Sigma^{\leq n-1}$ we will slightly abuse notation and write $\texttt{Ham}(\texttt{pwd}_1, \texttt{pwd}_2)$ instead of $\texttt{Ham}(\texttt{Pad}(\texttt{pwd}_1), \texttt{Pad}(\texttt{pwd}_2))$.

2.2 Shamir Secret Sharing

Let $\mathbb{F}$ be a finite field of size $|\mathbb{F}| \geq n + 1$. The Shamir Secret Sharing Scheme [Sha79] over a field $\mathbb{F}$ consists of three algorithms $(\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{SecretRecover})$. The algorithm $\mathsf{Setup}(n, t)$ takes n and input, picks n distinct points $x_1, \ldots, x_n \in \mathbb{F} \setminus \{0\}$ from the finite field $\mathbb{F}$ and outputs public parameters $\mathsf{pp} = ((n, t), x_1, \ldots, x_n)$. The share generation algorithm $\mathsf{ShareGen}(s, \mathsf{pp})$ takes as input a secret $s \in \mathbb{F}$ along with the public parameters pp and (1) sets $a_0 = s$, (2) picks coefficients $a_1, \ldots, a_{t-1} \in_R \mathbb{F}$ uniformly at random, (3) defines the polynomial $p(x) = \sum_{i=0}^{t-1} a_i x^i$, (4) computes $[\![s]\!]_i = p(x_i)$ for all $1 \leq i \leq n$, and (5) outputs the shares $[\![s]\!]_i, \ldots, [\![s]\!]_n$.

For any subset $S \subseteq [n]$ of $|S| = t$ shares we can use polynomial interpolation to recover the secret s given all of the shares $[\![s]\!]_j$ for each $j \in S$. In particular, the algorithm $\mathsf{SecretRecover}(S, [\![s]\!]_S)$ will first use polynomial interpolation to recover the polynomial $p(x)$ and then output $s = p(0)$. Thus, t shares suffice to recover the secret while for any subset of $|S| < t$ shares the vector $[\![s]\!]_S$ is uniformly random over $\mathbb{F}^{|S|}$.

In a bit more detail, for any $S \subseteq [n]$ of size $|S| = t$ we can interpolate the polynomial $p(x) = \sum_{i \in S} [\![s]\!]_i L_{i,S}(x)$ where $L_{i,S}(x) = \prod_{j \in S \setminus \{i\}} \left(\frac{x - x_i}{x_j - x_i}\right)$ denotes the Lagrange interpolating polynomial which can be computed based on the public parameters pp. We can also more directly compute $s = p(0) = \sum_{i \in S} [\![s]\!]_i L_{i,S}(0)$ where $L_{i,S}(0) = \prod_{j \in S \setminus \{i\}} \left(\frac{x_i}{x_i - x_j}\right)$. In particular, the secret s can be expressed as a linear sum over the shares $[\![s]\!]_j$ with coefficients $L_{j,S}(0)$ that can be calculated entirely from the public parameters.

3 Random Robust Secret Sharing

In this section, we will introduce the novel cryptographic notion of t out of n Random Robust Secret sharing with $t-1$-perfect privacy. Intuitively, $t-1$-perfect privacy means that any subset of $t-1$ shares are uniformly distributed. Random robustness means that we can (whp) *efficiently* recover the secret even if up to $n-t$ of the given n shares were *randomly* corrupted. By only requiring robustness against random corruptions we are able to overcome prior barriers and eliminate the robustness/privacy gap inherent in prior work [DHP+18, BFH+23]. We begin by defining Random Robust Secret Sharing in Section 3.1 and then in Section 3.2 we provide a construction. Finally, we prove that our construction satisfies t-random robustness and $t-1$-perfect privacy in Section 3.3.

3.1 Definition of Random Robust Secret Sharing

We begin by formally defining a Random Robust Secret Sharing (RRSS) scheme $\Pi_{\mathsf{RRSS}} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$. Intuitively, the new algorithm $\mathsf{Identify}$ is used to identify which shares were/were not randomly corrupted before we run $\mathsf{SecretRecover}$.

Before giving the formal definition, it will be helpful to introduce some notation. Given a set $\mathbb{S}$, a vector $\vec{c} = (c_1, \ldots, c_n) \in \mathbb{S}^n$ of dimension n over $\mathbb{S}$ and a subset $S \subseteq [n]$ of indices, it will be useful to define $\mathcal{D}(\mathbb{S}, \vec{c}, S) \doteq \left\{\vec{c'} \in \mathbb{S}^n \ : \ \vec{c'}_S = \vec{c}_S\right\}$ as the set of all vectors $\vec{c'} = (c'_1, \ldots, c'_n) \in \mathbb{S}^n$ which are consistent with $\vec{c}$ over the indices in S i.e., such that $c'_i = c_i$ for all $i \in S$. Intuitively, if $\vec{c}$ denotes a vector of n shares then sampling $\vec{c'}$ uniformly at random from $\mathcal{D}(\mathbb{S}_\lambda, c, S)$ is equivalent to replacing c_i with a uniformly random sample $c'_i \in_R \mathbb{S}_\lambda$ for each $i \notin S$. t-random robustness tells us that if $|S| \geq t$ then (whp) we should still be able identify t uncorrupted shares and recover the secret.

Definition 1 *Let $\lambda \in \mathbb{N}$ be the security parameter, p a polynomial, $\mathbb{D}_\lambda = \{0,1\}^\lambda$ be the λ-bit secret space, $n, t \in \mathbb{N}$ with $t < n$, PP set of all public parameters and $\mathbb{S}_\lambda$ the space of all possible shares. An $(n, t, \epsilon(\lambda))$-Random Robust secret sharing scheme Π_{RRSS} contains four (possibly randomized) polynomial-time algorithms: setup $\mathsf{Setup} : 1^\lambda \times \mathbb{N} \times \mathbb{N} \to PP$, share generation $\mathsf{ShareGen} : \mathbb{D}_\lambda \times PP \to \mathbb{S}_\lambda^n$, reconstruction $\mathsf{SecretRecover} : 2^{[n]} \times \mathbb{S}_\lambda^t \times PP \to \mathbb{D}_\lambda$ and valid share identifier $\mathsf{Identify} : \mathbb{S}_\lambda^n \times PP \to 2^{[n]}$ with the following properties:*

- *Correctness: For any secret $s \in \mathbb{D}_\lambda$, any pp in the support of $\mathsf{Setup}(1^\lambda, n, t)$, any $\vec{c} = ([\![c]\!]_1, \ldots, [\![c]\!]_n)$ in the support of $\mathsf{ShareGen}(s, \mathsf{pp})$, and any subset $S \subseteq [n]$ with $|S| \geq t$ we have $\mathsf{SecretRecover}(S, \vec{c}_S, \mathsf{pp}) = s$. Intuitively, this property says that we can always recover the original secret if we are given t shares.*
- *$(t-1)$-Perfect Privacy: for all secrets $s \in \mathbb{D}_\lambda$, all pp in the support of $\mathsf{Setup}(1^\lambda, n, t)$, all $S \subseteq [n]$ with $|S| \leq t-1$ and all $c' \in \mathbb{S}_\lambda^{|S|}$ we have $\Pr[\vec{c}_S = c'] = |\mathbb{S}_\lambda|^{-|S|}$ where $\vec{c} = ([\![c]\!]_1, \ldots, [\![c]\!]_n) \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$ and the randomness is taken over the random coins of $\mathsf{ShareGen}$. Intuitively, for any subset $S \subseteq [n]$ of $|S| \leq t-1$ shares the joint distribution over those shares is uniformly random over $\mathbb{S}_\lambda^{|S|}$.*
- *t-Random Robustness: for any $s \in \mathbb{D}_\lambda$, $S \subseteq [n]$ with $|S| \geq t$, any pp in the support of $\mathsf{Setup}(1^\lambda, n, t)$, and any $c = ([\![c]\!]_1, \ldots, [\![c]\!]_n)$ in the support of $\mathsf{ShareGen}(s, \mathsf{pp})$ we have*

$$\Pr_{\vec{c'} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}, S)} \left[S' = \mathsf{Identify}\left(\vec{c'}, \mathsf{pp}\right) \subseteq S \wedge |S'| \geq t\right] \geq 1 - \epsilon(\lambda) \ ,$$

where the randomness is only taken over the uniformly random choice of $[\![c']\!]_i$ *for each* $i \notin S$*. Note that we must have* $\vec{c'}_i = \vec{c}_i$ *for all* $i \in S$*. Intuitively, this property tells us that if at most* $n - t$ *shares are randomly corrupted while the remaining shares are correct then (whp)* Identify *will correctly identify* t *uncorrupted shares without missclassifying any corrupted shares.*

Remark 1 *The correctness, perfect privacy, and random robustness guarantees above hold for every public parameter* pp *in the support of* $\mathsf{Setup}(1^\lambda, n, t)$*; the same is true of our definition of security against malicious dealers. Thus, the security of the scheme does not depend on a trusted setup procedure or on any secret setup information, provided that parties can verify that* pp *is a valid output of* Setup*. In our construction,* Setup *outputs a vector* $\vec{a} = (a_1, \ldots, a_{3n}) \in_R \mathbb{F}_{2^{\lambda_1}}^{3n} \setminus \{0\}$ *so every party can independently verify* pp *by simply checking that each component of the vector is non-zero i.e.,* $a_j \neq 0$ *for each* $j \leq 3n$*. More generally, any randomized setup algorithm can be made publicly verifiable by including its random coins in the public parameters: define* $\mathsf{Setup}'(1^\lambda, n, t; r)$ *to compute* $\mathsf{pp} = \mathsf{Setup}(1^\lambda, n, t; r)$ *and output* $\mathsf{pp}' = (\mathsf{pp}, r)$*. Any party can then verify that* pp *was generated by* Setup *using the disclosed randomness* r*. Since our guarantees hold for every valid choice of* pp *in the support of* Setup*, they continue to hold even if the randomness* r *is chosen adversarially.*

Given a vector $\vec{c'} = ([\![c']\!]_1, \ldots, [\![c']\!]_n)$ of (possibly corrupted) shares it will often be convenient to write $\mathsf{SecretRecover}(\vec{c'}, \mathsf{pp})$ to be the algorithm that runs $S' = \mathsf{Identify}\left(\vec{c'}, \mathsf{pp}\right)$ and then outputs $\mathsf{SecretRecover}(S', \vec{c'}, \mathsf{pp})$ if $|S'| \geq t$; otherwise $\mathsf{SecretRecover}(\vec{c'}, \mathsf{pp})$ outputs $\perp$.

3.1.1 Security Against Malicious Dealers

It will also be useful to define an additional (optional) security guarantee against malicious dealers which ensures that if the i'th share is randomly corrupted, then the index i will not be output by Identify except with negligible probability — even if the attacker picked the initial shares maliciously with the goal of tricking the Identify algorithm. Similarly, if at least $n - t + 1$ shares are uniformly random (and the remaining shares are fixed a priori by the malicious dealer) then (whp) the algorithm Identify will recognize this and output $\perp$. More formally, we say that Identify achieves $(t-1, \epsilon(\lambda))$-Security against malicious dealers if the following two properties hold: (1) for any $\vec{c'} = (c'_1, \ldots, c'_n) \in \mathbb{S}_\lambda^n$, any pp in the support of $\mathsf{Setup}(1^\lambda, n, t)$, and any $S \subseteq [n]$ with $|S| \geq t$ we have

$$\Pr_{\vec{c''} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c'}, S)} \left[\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \neq \perp \wedge \mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \not\subseteq S\right] \leq \epsilon(\lambda) \ ,$$

and (2) for any $\vec{c'} = (c'_1, \ldots, c'_n) \in \mathbb{S}_\lambda^n$, any pp in the support of $\mathsf{Setup}(1^\lambda, n, t)$, and any $S \subseteq [n]$ with $|S| \leq t$ we have

$$\Pr_{\vec{c''} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c'}, S)} \left[\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) = \perp\right] \geq 1 - \epsilon(\lambda) \ .$$

In both cases the randomness is only taken over the uniformly random choice of $[\![c'']\!]_i$ for each $i \notin S$. Note that when we sample $\vec{c''}$ we will have $\vec{c''}_i = \vec{c'}_i$ for all $i \in S$.

While this final (optional) property is not required it is a useful guarantee, and the construction we give in Section 3.2 will satisfy this property. We provide alternate game-based security definitions for Random Robust Secret Sharing in Appendix D. However, these definitions are implied by the current ones.

3.2 Construction of Random Robust Secret Sharing

We now describe our $(n, t, 2^{-n\lambda_1})$-Random Robust Secret Sharing $\Pi_{\mathsf{RRSS}} = (\mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$. We reference our construction as **Construction 1** in Appendix I. However, **Construction 1** is described completely in the text below. Before giving the formal construction, we first provide a high-level description.

3.2.1 High level intuition of Π_{RRSS}.

Intuitively, we use Shamir secret sharing as the primary building block. In the setup phase, we setup the parameters of our Shamir secret sharing and select a set of $3n$ non-zero values $(a_1, \ldots, a_{3n}) \in (\mathbb{F}_{2^{\lambda_1}} \setminus \{0\})^{3n}$ which are part of our public parameter pp. Each individual share $[\![z]\!]_i$ of our secret s will consist of a vector of $6n+1$ Shamir Secret shares i.e., $[\![z]\!]_i = \left([\![s]\!]_i, [\![a]\!]^1_{2i-1}, \ldots, [\![a]\!]^{3n}_{2i-1}, [\![a]\!]^1_{2i}, \ldots, [\![a]\!]^{3n}_{2i}\right)$. The first component $[\![s]\!]_i$ is secret share of s generated using the regular t out of n Shamir Secret Sharing procedure i.e., $([\![s]\!]_1, \ldots, [\![s]\!]_n) \leftarrow \Pi^{\lambda}_{SSS,n,t}.\mathsf{ShareGen}_{2^\lambda}(n, t, s)$. This is the only component of $[\![z]\!]_i$ that is directly related to the secret s. The remaining Shamir Secret Shares are simply used to identify shares that have been randomly corrupted. In particular, for each $j \leq 3n$ the shares $[\![a]\!]^j_{2i-1}$ and $[\![a]\!]^j_{2i}$ are both Shamir Secrets of the public value a_j generated using a $2t$ out of $2n$ secret sharing scheme i.e., $[\![a]\!]^j_1, \ldots, [\![a]\!]^j_{2n} \leftarrow \Pi^{\lambda_1}_{SSS,2n,2t}.\mathsf{ShareGen}_{2^{\lambda_1}}(2n, 2t, a_j)$ for all $j \in [3n]$.

The shares $[\![a]\!]^j_i$ are useful to identify a set $S \subseteq [n]$ of at least $|S| \geq t$ uncorrupted indices using Gaussian Elimination. To provide some intuition assume that we are given shares $[\![z']\!]_1, \ldots, [\![z']\!]_n$ and that t of these shares are correct while $n-t$ have been randomly corrupted. Let $S \subseteq [n]$ with $|S| = t$ denote the (currently unknown) set of valid share indexes such that $[\![z']\!]_i = [\![z]\!]_i$ for all $i \in S$. Given a subset $S \subseteq [n]$ of the shares of our secret s it will be convenient to write $\hat{S} = \bigcup_{i \in S}\{2i-1, 2i\}$ to denote the corresponding subset of secret shares for each public valid a_j. Applying Lagrange interpolation we have $a_j = \sum_{i \in S}\left(L_{2i-1,\hat{S}}(0)[\![a]\!]^j_{2i-1} + L_{2i,\hat{S}}(0)[\![a]\!]^j_{2i}\right)$ for each $j \leq 3n$. Observe that each term $L_{2i-1,\hat{S}}(0)$ and $L_{2i,\hat{S}}(0)$ is non-zero and entirely independent of j and of the secret s. However, we cannot directly compute $L_{2i-1,\hat{S}}(0)$ since it does depend on the subset $S \subseteq [n]$ of correct shares which is currently unknown. Instead we introduce $2n$ variables x_i for each $i \leq 2n$ and define an (over-constrained) system of $3n$ linear equations where for each $j \leq 3n$ we have $a_j = \sum_{i=1}^{2n} x_i [\![a']\!]^j_i$.

Intuitively, if $S \subseteq [n]$ is a set of t uncorrupted indices then the system has a valid solution by setting $x_{2i-1} = L_{2i-1,\hat{S}}(0)$ and $x_{2i} = L_{2i,\hat{S}}(0)$ for each $i \in S$ and setting $x_{2i-1} = x_{2i} = 0$ for each $i \notin S$. Thus, our approach is to solve the system of $3n$ linear equations for our $2n$ unknown variables. Once we solve for $x_1, \ldots, x_{2n}$ we build our set $S \subseteq [n]$ of uncorrupted shares by setting $S = \{i : x_{2i} > 0 \vee x_{2i-1} \geq 0\}$. Because the system is over-constrained ($3n$ equations vs $2n$ variables) we can argue that (whp) we must have $x_{2i-1} = x_{2i} = 0$ for all shares $i \in [n] \setminus S$ that were randomly corrupted. Similarly, we can argue that (whp) we will have $|\{i : x_i \neq 0\}| \geq 2t-1$ for every feasible solution. Thus, if there are t valid shares we will (whp) find a set S containing $|S| \geq t$ shares and if there are fewer than t valid shares there will be no feasible solution (whp).

3.2.2 Full Description

- **Setup** $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, t)$: Given the security parameter λ we select $\lambda_1 \geq \max\left\{\lceil \log_2 2n \rceil, \lceil \frac{\lambda}{n} \rceil\right\}$.[1] Let $\Pi^{\lambda}_{SSS,n,t} = (\mathsf{ShareGen}_{2^\lambda}, \mathsf{SecretRecover}_{2^\lambda})$ (resp. $\Pi^{\lambda_1}_{SSS,2n,2t} = (\mathsf{ShareGen}_{2^{\lambda_1}}, \mathsf{SecretRecover}_{2^{\lambda_1}})$) be a secure (n,t) (resp. $(2n,2t)$ Shamir secret sharing over filed $\mathbb{F}_{2^\lambda}$ (resp. $\mathbb{F}_{2^{\lambda_1}}$). Let $\vec{a} = (a_1, \ldots, a_{3n}) \in_R (\mathbb{F}_{2^{\lambda_1}} \setminus \{0\})^{3n}$ be $3n$ non-all-zero vector whose elements are chosen from the small size finite field $\mathbb{F}_{2^{\lambda_1}}$. Return the public parameter $\mathsf{pp} = (\vec{a}, n, t, \Pi^{\lambda}_{SSS,n,t}, \Pi^{\lambda_1}_{SSS,2n,2t})$.
- **Share Generation** $([\![z]\!]_1, \ldots, [\![z]\!]_n) = \mathsf{ShareGen}(s, \mathsf{pp})$: This algorithm takes as input the secret s and public parameters pp (specifying t and n) and generates n shares $[\![z]\!]_i$ for all $i \in [n]$ as follows:
  (1) Parse: $\vec{a} = (a_1, \ldots, a_{3n}) \in \mathsf{pp}$.
  (2) $([\![s]\!]_1, \ldots, [\![s]\!]_n) \leftarrow \Pi^{\lambda}_{SSS,n,t}.\mathsf{ShareGen}_{2^\lambda}(n, t, s)$
  (3) $\forall j \in [3n]$: compute $[\![a]\!]^j_1, \ldots, [\![a]\!]^j_{2n} \leftarrow \Pi^{\lambda_1}_{SSS,2n,2t}.\mathsf{ShareGen}_{2^{\lambda_1}}(2n, 2t, a_j)$.

[1]. We used $\lambda_1 \in \{8, 9, 16\}$ in our empirical evaluations.

(4) $\forall i \in [n]$: Set $[\![z]\!]_i = \left([\![s]\!]_i, [\![a]\!]^1_{2i-1}, \ldots, [\![a]\!]^{3n}_{2i-1}, [\![a]\!]^1_{2i}, \ldots, [\![a]\!]^{3n}_{2i}\right)$.

(5) Return $(([\![z]\!]_1 \ldots, [\![z]\!]_n))$

- **Recovery from Valid Shares:** $s \leftarrow \mathsf{SecretRecover}\,(S, \vec{z}_S)$: This algorithm takes as input a set $S \subseteq [n]$ of size $|S| \geq t$ and $|S|$ shares $\vec{z}_S$ and recovers the secret s as follows:

  (1) Parse $\vec{z}_S$ to recover $[\![z]\!]_i$ for each $i \in S$.

  (2) For each $i \in S$ extract $[\![s]\!]_i$ from $[\![z]\!]_i = \left([\![s]\!]_i, [\![a]\!]^1_{2i-1}, \ldots, [\![a]\!]^{3n}_{2i-1}, [\![a]\!]^1_{2i}, \ldots, [\![a]\!]^{3n}_{2i}\right)$. Let $\vec{s}_S = ([\![s]\!]_i)_{i \in S}$.

  (3) Return $s = \Pi^{\lambda}_{SSS,n,t}.\mathsf{SecretRecover}_{2^\lambda}\,(S, \vec{s}_S)$.

- **Valid Share Identification** $S \leftarrow \mathsf{Identify}\,(([\![z']\!]_1 \ldots, [\![z']\!]_n), \mathsf{pp})$: This algorithm takes the input shares $([\![z']\!]_1 \ldots, [\![z']\!]_n)$ and if it contains at most $n - t$ randomly corrupted shares, it returns a set $S \subseteq [n]$ identifying the indexes of at least t valid shares (whp). The share identification algorithms is described in the following steps:

  (1) Parse: $\vec{a} = (a_1, \ldots, a_{3n}) \in \mathsf{pp}$

  (2) for all $i \in [n]$ parse: $[\![z']\!]_i = \left([\![s']\!]_i, [\![a']\!]^1_{2i-1}, \ldots, [\![a']\!]^{3n}_{2i-1}, [\![a']\!]^1_{2i}, \ldots, [\![a']\!]^{3n}_{2i}\right)$

  (3) Form the system of linear equations $(\vec{a})^T = AX^T$ in which $A = \begin{bmatrix} a_{1,1} & \ldots & a_{1,2n} \\ \ldots & a_{i,j} & \ldots \\ a_{3n,1} & \ldots & a_{3n,2n} \end{bmatrix}$ where $a_{i,j} = [\![a']\!]^i_j$ and $X = (x_1, \ldots, x_{2n})$ is a $2n$ unknown variable vector.

  (4) Solve $\vec{a} = AX$ to find the values $x_1, \ldots, x_{2n}$ and set $S = \{i \in [n] : x_{2i-1} \neq 0 \vee x_{2i} \neq 0\}$. If there are no solutions then return $\perp$.

  (5) If $|S| < t$, then return $\perp$; otherwise, return S as the set of valid shares.

- **Valid Share Identification and Secret Reconstruction** $\{s, \perp\} := \mathsf{SecretRecover}\,(([\![z']\!]_1 \ldots, [\![z']\!]_n), \mathsf{pp})$: This algorithm takes the input shares $\vec{z'} = ([\![z']\!]_1 \ldots, [\![z']\!]_n)$ and if we have at least t valid shares, it returns the secret s; otherwise, it returns $\perp$. The reconstruction secret is described as follows.

  (1) Run $S \leftarrow \mathsf{Identify}\,(([\![z']\!]_1 \ldots, [\![z']\!]_n), \mathsf{pp})$ to identify the set of valid shares.

  (2) If $S = \perp$ then return $\perp$.

  (3) If $|S| > t$ discard the last $|S| - t$ items from S to ensure that $|S| = t$.

  (4) For each $i \in S$ extract $[\![s']\!]_i$ from $[\![z']\!]_i = \left([\![s']\!]_i, [\![a']\!]^1_{2i-1}, \ldots, [\![a']\!]^{3n}_{2i-1}, [\![a']\!]^1_{2i}, \ldots, [\![a']\!]^{3n}_{2i}\right)$. Let $\vec{s'}_S = ([\![s']\!]_i)_{i \in S}$.

  (5) Return $s = \Pi^{\lambda}_{SSS,n,t}.\mathsf{SecretRecover}_{2^\lambda}\left(S, \vec{s'}_S\right)$.

Share Size and Running Time: Each share $[\![z]\!]_i$ requires $\lambda + 6n\lambda_1$ bits to encode since there are $6n$ field elements from $\mathbb{F}_{2^{\lambda_1}}$ and a single field element from $\mathbb{F}_{2^\lambda}$. Thus, the total amount of bits to encode all n shares is $n\lambda + 6n^2\lambda_1$. From a computational efficiency standpoint the most complex operation is solving the system of linear equations in $\mathsf{Identify}$. In particular, the time complexity is dominated by solving system of linear equations of size $O(n) \times O(n)$, i.e., $O(n^\omega)$ where $\omega \leq 2.3716$ [WXXZ24].

Remark 2 *We note that* Π_{RRSS} *is not secure against* adversarial corruptions. *For example, an attacker could take the share* $[\![z]\!]_i = \left([\![s]\!]_i, [\![a]\!]^1_{2i-1}, \ldots, [\![a]\!]^{3n}_{2i-1}, [\![a]\!]^1_{2i}, \ldots, [\![a]\!]^{3n}_{2i}\right)$ *and simply modify the value of* $[\![s]\!]_i$

to obtain $[\![z']\!]_i = \left([\![s']\!]_i, [\![a]\!]^1_{2i-1}, \ldots, [\![a]\!]^{3n}_{2i-1}, [\![a]\!]^1_{2i}, \ldots, [\![a]\!]^{3n}_{2i}\right)$. *Because this first component is never used to identify (in)valid shares the modification will go undetected, and will likely result in the recovery of an incorrect secret* $s' \neq s$. *However, this is not a problem in our context because a random corruption of* $[\![z]\!]_i$ *will randomly modify each individual component* $[\![a]\!]^j_{2i-1}$ *and* $[\![a]\!]^j_{2i}$ *for each* $j \leq 3n$.

Remark 3 *It is natural to wonder why we use* $2t$ *out of* $2n$ *Shamir secret sharing instead of* t *out of* n *secret sharing which seems to be more natural and efficient. The reason is because we made the value* a_j *public for each* $j \leq 3n$. *Effectively* a_j *becomes a "bonus share" making it possible for the attacker to interpolate the Shamir polynomial with one fewer share. Suppose that we used* t *out of* n *secret sharing and let* $p_j(x)$ *be the degree* $t-1$ *polynomial we picked when we generate* n *shares* $[\![a]\!]^j_1, \ldots [\![a]\!]^j_n$ *of* a^j *using Shamir Secret Sharing. Normally, we would need* t *shares to interpolate* p_j. *However, because* $p_j(0)$ *is known a priori an attacker could actually interpolate* p_j *using just* $t-1$ *shares.*

We instead use $2t$ *out of* $2n$ *secret sharing to prevent an attacker who has just* $t-1$ *shares from interpolating the polynomial* $p_j(x)$. *The polynomial* $p_j(x)$ *now has degree* $2t-2$ *and each share* $[\![z]\!]_i$ *contains exactly two shares of* a^j. *Given* $t-1$ *shares* $[\![z]\!]_i$ *we can only obtain* $2t-2$ *shares of* a^j. *Even with the "bonus share"* $p_j(0) = a^j$ *this will be insufficient to interpolate* p_j. *Thus, the* $2t-2$ *shares of* a^j *will still be distributed uniformly at random.*

3.3 Security proof of Π_{RRSS}

Theorem 4 *The construction* $\Pi_{\mathsf{RRSS}} = (\mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$ *described in* ***Construction 1*** *is a* $(n, t, \epsilon(\lambda))$*-Random Robust Secret Sharing scheme with* $\epsilon(\lambda) \leq 2^{-\lambda}$. Π_{RRSS} *also achieves* $(t-1, \epsilon(\lambda))$*-Security against malicious dealers.*

Proof of Theorem 4: (Intuition) The full proof of Theorem 4 is deferred to Appendix E.1. We sketch the intuitive ideas here.

It is straightforward to argue that any subset $S \subseteq [n]$ of $|S| \leq t-1$ shares of $[\![z]\!]_1, \ldots, [\![z]\!]_n$ are distributed uniformly at random in $\left(\mathbb{F}_{2^\lambda} \times \mathbb{F}^{6n}_{2^{\lambda_1}}\right)^{|S|}$. Intuitively, this follows directly from $(t-1)$-perfect privacy of the underlying Shamir secret sharing scheme since each individual component of the share $[\![z]\!]_i = \left([\![s]\!]_i, [\![a]\!]^1_{2i-1}, \ldots, [\![a]\!]^{3n}_{2i-1}, [\![a]\!]^1_{2i}, \ldots, [\![a]\!]^{3n}_{2i}\right)$ comes from a Shamir Secret Sharing Scheme.

t-Random Robustness follows intuitively because there are $3n$ linear constraints and $2n$ variables. Suppose that the first share $[\![z]\!]_1$ is randomly corrupted to $[\![z']\!]_1 = \left([\![s']\!]_i, [\![a']\!]^1_1, \ldots, [\![a']\!]^{3n}_1, [\![a']\!]^1_2, \ldots, [\![a']\!]^{3n}_2\right)$. Now consider an a priori fixed variable assignment where $x_1 \neq 0$ (or equivalently $x_2 \neq 0$). Because each $[\![a']\!]^j_1$ is uniformly random and $x_1 \neq 0$ we can view each term $x_1[\![a']\!]^j_1$ as uniformly random for each $j \leq 3n$. This means that the probability that each individual constraint is satisfied by our a priori fixed assignment is at most $2^{-\lambda_1}$ and the probability that *all* $3n$ constraints are satisfied is at most $2^{-3n\lambda_1}$. We can now union bound over all $2^{2n\lambda_1}$ possible variable assignments to conclude that, except with probability $2^{-n\lambda_1}$ we will have $x_{2i-1} = x_{2i} = 0$ for all shares $[\![z]\!]_1$ that are randomly corrupted. We note that the same exact argument applies to show that the scheme achieves security against malicious dealers. It doesn't matter whether or not the initial shares $[\![z]\!]_1, \ldots, [\![z]\!]_n$ were generated honestly or not. If $[\![z']\!]_1$ is randomly corrupted and $x_1 \neq 0$ then each term $x_1[\![a']\!]^j_1$ is still uniformly random so that the probability each individual constraint is satisfied is at most $2^{-\lambda_1}$ and the probability all $3n$ constraints are satisfied is at most $2^{-3n\lambda_1}$. We can now union bound over all $2^{2n\lambda_1}$ possible variable assignments to conclude that, except with probability $2^{-n\lambda_1}$ we will have $x_{2i-1} = x_{2i} = 0$ for all shares $[\![z]\!]_1$ that are randomly corrupted which implies that i will not be in the set of indices returned by $\mathsf{SecretRecover}$. If we assume that at least $n-t+1$ shares are corrupted this implies that, except with probability $2^{-n\lambda_1}$, we will have $|\{i : x_{2i} \neq 0 \vee x_{2i-1} \neq 0\}| < t$ in which case $\mathsf{Identify}$ will output $\perp$ as required. We note that $2^{-\lambda_1 n} \leq 2^{-(\lambda/n)n} = 2^{-\lambda}$.

A similar argument shows that, for any a priori fixed variable assignment with $|\{i : x_i \neq 0\}| < 2t-1$, the

probability that the assignment is consistent with all $3n$ constraints is at most $2^{-3n\lambda_1}$. The primary difference is that we are no longer assuming that any shares are randomly corrupted. However, any subset of $2t-2$ shares from a $2t$ out of $2n$ Shamir Secret Sharing Scheme can be viewed as uniformly random so we can apply the same reasoning. We can once union bound over all $2^{2n\lambda_1}$ possible variable assignments to conclude that, except with probability $2^{-n\lambda_1}$, no assignment with $|\{i : x_i \neq 0\}| < 2t-1$ will satisfy all of our equations. This implies that the set $S = \{i \ : \ x_{2i-1} \neq 0 \vee x_{2i} \neq 0\}$ returned by $\mathsf{Identify}$ must have size at least t. In particular, as long there are at least t valid shares to identify we will find at least t valid shares (whp). In particular, if $S \subseteq [n]$ is a set of $|S| \geq t$ uncorrupted shares then we can always obtain a feasible assignment by setting $x_{2i-1} = L_{2i-1,S}(0)$ and $x_{2i} = L_{2i,S}(0)$.

□

4 Fuzzy Password-based Authenticated Key Exchange

As our first application of Random Robust Secret Sharing, we show how to use RRSS to construct a secure and efficient Fuzzy Password-based Authenticated Key Exchange (fPAKE) protocol. A fPAKE protocol enables two parties holding noisy versions of a shared low-entropy password to securely establish a session key over an unauthenticated channel.

Our construction is obtained by modifying the protocol of [BFH+23] to use RRSS. In the remainder of this section, we first present the ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$. Our definition closely follows [DHP+18], but without weakening the functionality. We then describe our construction and prove that it UC-realizes $\mathcal{F}_{\mathsf{fPAKE}}$.

4.1 fPAKE Ideal functionality

Due to space limitations defer the full/formal description of the ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$ to Appendix F.1 — see Figure 3. Intuitively, the ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$ models a fuzzy password-authenticated key exchange between two parties, Party_0 and Party_1, in the presence of an adversary. A party Party_i can initiate the fPAKE protocol by sending a NewSession query to the ideal functionality along with a session identifier sid and a password $\mathtt{pwd}_i$. Suppose that the other party submits a password $\mathtt{pwd}_{1-i}$ and that the adversary does not block message delivery between the honest parties and the ideal functionality. In this case, the ideal function will deliver the same uniformly random key K' to both parties as long as the Hamming Distance $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}_{1-i}) \leq \ell$ is sufficiently small. On the other hand if $\mathtt{Ham}(\mathtt{pwd}, \mathtt{pwd}') > \ell$ then the ideal functionality will send independently sampled keys K_0 and K_1 to Party_0 and Party_1 respectively.

To model an active attacker (e.g., Man-in-the-Middle) the ideal functionality allows the adversary to submit a special TestPassword query with a guess $\mathtt{pwd}^*$ for Party_i's password in session sid. However, the attacker can only submit at most one guess $\mathtt{pwd}^*$ for each registered tuple $(\mathsf{sid}, \mathsf{Party}_i)$ i.e., a man-in-the-middle attacker may make at most one guess at Party_0's password and one guess at Party_1's password per fPAKE session. Assuming that query is fresh, the outcome of the query $(\text{TestPassword}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}^*)$ query is either (1) WrongGuess, if $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}^*) > \ell$, or (2) $(\text{CorrectGuess}, L_c(\mathtt{pwd}_i, \mathtt{pwd}^*))$ if $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}^*) \leq \ell$. For our protocol we use the leakage function $L_c(\mathtt{pwd}_i, \mathtt{pwd}^*) = \{j : \mathtt{pwd}_i[j] = \mathtt{pwd}^*[j]\}$ i.e., if the adversary's guess is close enough the ideal functionality may reveal the locations of indices where the password guess $\mathtt{pwd}^*$ matches $\mathtt{pwd}_i$.

The ideal fPAKE functionality we consider is essentially the same as in [DHP+18] with one key difference. In [DHP+18] the outcome of a test password query $(\text{TestPassword}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}^*)$ will be CorrectGuess if $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}^*) \leq \ell'$ for some parameter $\ell' > \ell$, while honest key-exchange will still only succeed if $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}_{1-i}) \leq \ell$. The gap arises due to the perfect privacy/robustness gap in their underlying secret sharing scheme. We are able to eliminate this gap using random robust secret sharing.

4.2 fPAKE Construction using RRSS

Our fPAKE protocol is described in Figure 1. The protocol closely follows fPAKE protocol of Bootle et al. [BFH+23][2]. The differences between our construction and [BFH+23] are highlighted in blue.For completeness we also include the original fPAKE construction of Bootle et al. [BFH+23] in Appendix I — see Figure 11. The primary difference is that we replace the random encoding scheme with our Random Robust Secret Sharing scheme[3].

As in Bootle et al.'s (relaxed) fPAKE protocol, we use an iPAKE protocol π as a building block. The iPAKE protocol π must UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{iPAKE}}$ [DHP+18, BFH+23]. For completeness we provide the full $\mathcal{F}_{\mathsf{iPAKE}}$ functionality in Figure 6 in Appendix G.3. n iPAKE is like a regular PAKE protocol except that one party (sender) picks the key K. Intuitively[4], the ideal functionality $\mathcal{F}_{\mathsf{iPAKE}}$ guarantees that if the password provided by the second party (receiver) matches the sender's password exactly then the receiver will receive the key K; otherwise the ideal functionality sends a uniformly random key K' (unrelated to K) to the receiver. As in [BFH+23] we also use the split authentication functionality $\mathcal{F}_{\mathsf{SA}}$ of Barak et al. [BCL+05], recalled in Figure 7 in Appendix G.4,o model the ability of an attacker to run two separate executions of an authentication scheme (one with each party) without the two parties realizing it. The attacker may choose to actively launch a man-in-the-middle attack and execute separate protocols with each party or the attacker may choose to passively observe the messages exchanged (delaying message delivery if/when s/he chooses). However, if the attacker chooses to actively launch a man-in-the-middle attack the $\mathcal{F}_{\mathsf{SA}}$ functionality prevents the attacker from forwarding messages between separate protocol instances i.e., the attacker must separately interact with each party. The functionality $\mathcal{F}_{\mathsf{SA}}$ can be realized using digital signatures without requiring a public-key infrastructure.

The key idea behind our construction (and [DHP+18, BFH+23]) is to execute the iPAKE protocol n times — one time for each character in the password. Intuitively, after the ith iPAKE execution the receiver will either receive the ith key K_i selected by the sender if the two passwords match on character i; otherwise the receiver will get a random unrelated key K'_i. After the iPAKE protocols are complete the sender will pick a fresh string s and generate n shares $[\![z]\!]_1, \ldots, [\![z]\!]_n$ of s using a $(n, n-\ell, \epsilon(\lambda))$-Random Robust Secret Sharing Scheme[5]. The sender will use each key K_i as a one-time-pad and send $E_1, \ldots, E_n$ to the receiver where $E_i = [\![z]\!]_i \oplus K_i$. Intuitively, if the ith character of the passwords matched matched then the receiver will recover $E_i \oplus K_i = [\![z]\!]_i$; and if the passwords did not match on the ith character then the receiver will recover $E_i \oplus K'_i = [\![z']\!]_i$ where $[\![z']\!]_i$ is uniformly random because K'_i is uniformly random. If the passwords matched on at least $n-\ell$ characters then, by $(n-\ell)$ random robustness, the receiver will be able to efficiently identify the correct shares and recover s. The final session key $K = \mathcal{H}((\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}), s)$ is obtained by hashing s along with the session ids sid and $\mathsf{sid}_{\mathbb{H}}$. By including the session id $\mathsf{sid}_{\mathbb{H}}$ from $\mathcal{F}_{\mathsf{SA}}$ we ensure that if the attacker tampers with any iPAKE execution then both parties will obtain separate/random keys. This is crucial to protect against an attacker who attempts to guess a single character of the password by actively interfering in the corresponding iPAKE execution — see Remark 20 in Appendix I.3 for additional discussion.

If the passwords match on fewer than $n - \ell$ characters then, by $n - \ell - 1$-perfect privacy, the receiver will only learn that the passwords were not sufficiently close. By $n - \ell - 1$ perfect privacy the correct shares

[2]The original (relaxed) fPAKE protocol of [DHP+18] contained a flaw that was later corrected in [BFH+23]; this issue was independent of the underlying secret sharing mechanism. The corrected protocol of [BFH+23] was proven UC-secure, but only with respect to a weakened ideal functionality that necessarily permits leakage when the Hamming Distance is at most t_2—even if it exceeds t_1, so that key exchange fails.

[3]The protocol of [BFH+23] also required an extra hash value which was necessary due to their use of list decoding during secret recovery — the extra hash value is "helper information" used to identify/validate the correct codeword from a list of possibilities. Because we do not perform list decoding this extra helper information is unnecessary for our protocol and can be eliminated which simplifies our security analysis.

[4]We are oversimplifying a bit. If the adversary guesses the password (or corrupts a participant) then the adversary is allowed to modify the session key K that is delivered. It is also possible that the adversary delays delivery of network messages indefinitely so that the receiver never actually receives K.

[5]We take advantage of the fact that our RRSS setup algorithm is deterministic to avoid sending the public parameters pp and avoid forcing the receiver to verify that pp was generated correctly.

(when the passwords matched at character i) will be perfectly indistinguishable from the uniformly random shares that were received (when the passwords did not match at character i). Thus, an attacker won't learn which characters of the password matched nor will the attacker learn how many characters were matched overall. Similarly, because the RRSS scheme is secure against malicious dealers a malicious sender will not be able to fool the receiver by generating fake shares $[\![z'']\!]_i$ — when the receiver runs the $\mathsf{Identify}$ algorithm it will still output $\perp$ with high probability because at least $\ell + 1$ shares will be randomly corrupted.

At the end of the protocol the sender does not know for certain whether or not the receiver learned the same key. However, this is not part of the protocol or the ideal functionality fPAKE. If desired the receiver could prove that it received the correct key K after the protocol e.g., by sending $\mathcal{H}(K)$ to the sender or by encrypting any message using an authenticated encryption scheme (AES-GCM) with the symmetric key K.

4.3 fPAKE Analysis

We first prove that our fPAKE is $\epsilon(\lambda)$-correct in Theorem F.1. Intuitively, this means that if the adversary is passive and the input passwords pwd_0 and pwd_1 for each party have Hamming Distance is at most ℓ then the probability that both parties output different keys is at most $\epsilon_{\mathsf{iPAKE}}(\lambda)$— see Definition 4 in Appendix F.1 for a formal definition. We assume that our hash function $\mathcal{H}$ is a random oracle which UC-realized the ideal functionality $\mathcal{F}_{\mathsf{GRO}}$ for a Global Random Oracle— see Figure 8 in Appendix G.5. Our proof is similar to [BFH+23, Thm 1] except that the protocol of [BFH+23] requires a helper hash value to help with list decoding and a hash collision could violate the correctness guarantee of the protocol. Our correctness bound for $\epsilon(\lambda)$ does not include a term for hash collisions as we do not require a helper hash value. However, correctness could break if the RRSS identification algorithm fails which happens with probability at most $\epsilon_{\mathsf{RRSS}}(\lambda)$.

Theorem 5 *(fPAKE Correctness) If Π_{RRSS} is as an $(n, n-\ell, \epsilon_{\mathsf{RRSS}}(\lambda))$-random robust secret sharing, π is $\epsilon_{\mathsf{iPAKE}}(\lambda)$-correct, the hash function $\mathcal{H}$ UC-realizes $\mathcal{F}_{\mathsf{GRO}}$ and UC-realizes $\mathcal{F}_{\mathsf{iPAKE}}$, then the protocol in Figure 1 is (ℓ, ϵ)-correct for $\epsilon(\lambda) = n\epsilon_{\mathsf{iPAKE}}(\lambda) + \epsilon_{\mathsf{RRSS}}(\lambda)$ where q upper bounds the total number of queries to $\mathcal{H}$.*

See Appendix F.1 for the formal proof of Theorem F.1.

Theorem 6 (fPAKE UC-Realization) *If Π_{RRSS} is an $(n, n-\ell, \epsilon_{\mathsf{RRSS}}(\lambda))$-random robust secret sharing which additionally provides $(n-\ell-1, \epsilon_{\mathsf{RRSS}}(\lambda))$-security against malicious dealers, the hash function $\mathcal{H}$ UC-realizes $\mathcal{F}_{\mathsf{GRO}}$, and π is $(t_{\mathsf{iPAKE}}(\lambda), \epsilon_{\mathsf{iPAKE}}(\lambda))$-universally compossible which realizes $\mathcal{F}_{\mathsf{iPAKE}}$ using simulator $\mathtt{Sim}_{\mathsf{iPAKE}}$ in presence of the environment $\mathcal{Z}^*$ running in time at most $t_{\mathsf{iPAKE}}(\lambda)$, then the protocol in Figure 1 UC-realizes $\mathcal{F}_{\mathsf{fPAKE}}$ in the $\mathcal{F}_{\mathsf{SA}}$-hybrid model with error tolerance in password ℓ and leakage threshold $\gamma = \ell$.*

The proof of Theorem 6 follows the proof of [BFH+23] very closely with nearly identical hybrids. For completeness we include the full proof in appendix Appendix F.2.

At a high level, we define a sequence of computationally indistinguishable hybrids. In the first hybrid, the environment interacts with the real-world execution of the protocol. In the final hybrid, the environment interacts with the ideal functionality together with a suitably defined simulator. The indistinguishability of consecutive hybrids implies that no malicious environment can distinguish between the real-world protocol and the ideal-world execution. Consequently, our protocol securely UC-realizes the target ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$.

In one of the crucial hybrid steps we invoke $(t-1)$-perfect privacy of our RRSS scheme to argue that, when the passwords match on at most $t-1 = n-\ell-1$-characters and the sender is honest, the simulator can replace *the entire* message $\mathbf{E} = ([\![z]\!]_1 \oplus K_1, \ldots, [\![z]\!]_n \oplus K_n)$ with a random string. Intuitively, if $J \subseteq [n]$ denotes the set of indices where the password's matched then we can invoke iPAKE security to replace $E_i = [\![z]\!]_i \oplus K_i$ with a uniformly random string E'_i for all $i \notin J$ i.e., the receivers key K'_i is uniformly random and uncorrelated with K_i so the receiver will not be able to distinbuish $E_i = [\![z]\!]_i \oplus K_i$ from E'_i. We can then invoke $(t-1)$-perfect privacy to replace E_j with a uniformly random string E'_j for the remaining $|J| \leq t-1$ shares i.e., for each $j \in J$ the string $E_j = [\![z]\!]_j \oplus K_j$ is indistinguishable from random because the share $[\![z]\!]_j$

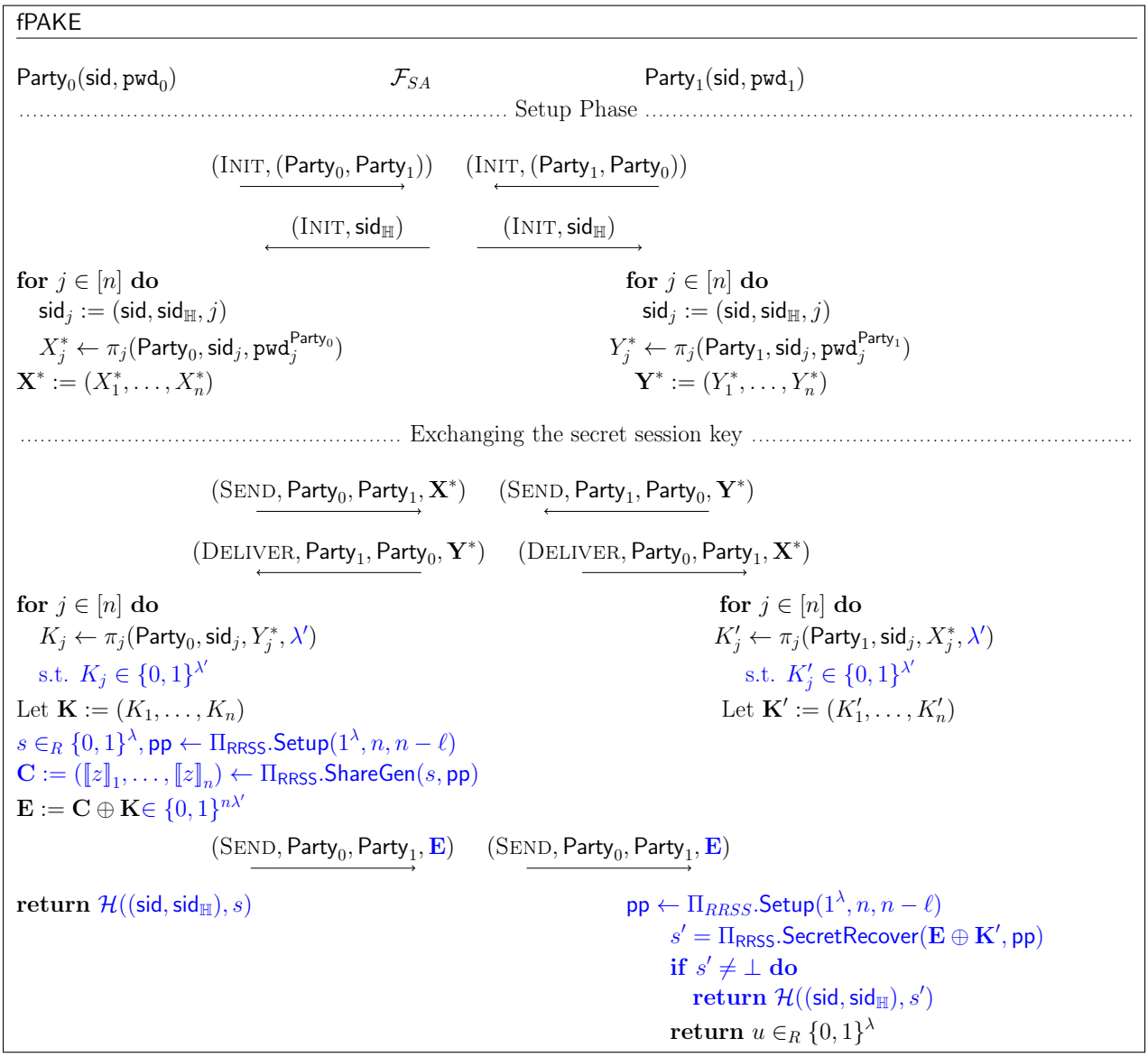


Figure 1: fPAKE protocol from our RRSS.

is uniformly random — even if the key K_j is known. This is the same hybrid argument as [BFH+23] except that they can only invoke $t_1 < n - \ell - 1$-perfect privacy so their simulator can only make the replacement if the passwords match on at most $t_1 < n - \ell - 1$ characters.

As a result, the final inputs to $\Pi_{\mathsf{RRSS}}.\mathsf{SecretRecover}$ are either (1) at least t-valid shares if the passwords are close which suffices to recover s under random corruptions, or (2) at most $t-1$-valid shares if the passwords are not close. In the second case $(t-1)$-perfect privacy ensures that all shares can be viewed as uniformly random and fully independent of the secret s.

An astute reader will observe that a malicious sender could pick arbitrary shares $(\llbracket z' \rrbracket_1, \ldots, \llbracket z' \rrbracket_n)$ and send $E'_i = \llbracket z' \rrbracket_i \oplus K_i$ to the receiver for each $i \in [n]$. If we let $J \subseteq [n]$ denote the indices where the two passwords match then the receiver will correctly recover the share $\llbracket z'' \rrbracket_j = E'_j \oplus K'_j = \llbracket z' \rrbracket'_j \oplus K'_j \oplus K'_j = \llbracket z' \rrbracket_j$ for each $j \in J$. By contrast, for each index $j \notin J$ the recovered share will still be a uniformly random string $\llbracket z'' \rrbracket_j = E'_j \oplus K'_j$ where K'_j is uniformly random due to iPAKE security. In our security proof we appeal to malicious-dealer security to argue that, except with negligible probability $\epsilon_{\mathsf{RRSS}}(\lambda)$, a malicious sender won't be able to fool an honest receiver. In particular, if $|J| < t$ then the $\mathsf{Identify}$ algorithm will output $\perp$ when it is executed by the honest receiver.

5 Conditional Encryption for Arbitrary Hamming Distance

As our second application, we use Random Robust Secret Sharing (RRSS) to build an efficient conditional encryption scheme for the Hamming-distance predicate supporting arbitrary thresholds. Conditional encryption, introduced by Ameri and Blocki [AB24] for TypTop [CWP+17], augments public-key encryption with a procedure $\mathsf{CEnc}_{pk}(c, m_1, m_2)$ that outputs a ciphertext c' which decrypts to m_2 if $\mathtt{Ham}(m, m_1) \leq \ell$, and otherwise reveals no information about m, m_1, or m_2—even given the secret key. This is formalized by requiring c' to be indistinguishable from a simulated ciphertext generated independently of (c, m_1, m_2) whenever $\mathtt{Ham}(m, m_1) > \ell$.

Our construction follows Ameri–Blocki [AB24] but replaces Shamir Secret sharing with RRSS. We provide an intuitive (slightly overly simplified) overview. Given Paillier ciphertexts $c[i] = (N+1)^{m[i]} R_i^N \bmod N^2$ for each individual character $m[i]$ of m, we sample a symmetric key K and generate n shares $\llbracket s \rrbracket_1, \ldots \llbracket s \rrbracket_n$ of K using an $(n, n-\ell, \epsilon(\lambda))$-RRSS scheme and then map each share $\llbracket s \rrbracket_i$ to a random integer $\hat{s}_i$ in $\mathbb{Z}_N$ subject to the constraint that $\llbracket s \rrbracket_i = \hat{s}_i \bmod 2^\lambda$. Using Paillier's homomorphic properties, we generate a conditional ciphertext of the form

$$c'[i] = (N+1)^{(m[i]-m_1[i])r_i + \hat{s}_i} z_i^N,$$

where $r_i \in \mathbb{Z}_N$ and $z_i \in \mathbb{Z}_N^*$ are fresh randomness. Intuitively, if $m[i] = m_1[i]$, then $c'[i]$ decrypts to $(m[i] - m_1[i])r_i + \hat{s}_i = \hat{s}_i$ allowing for the recovery of the correct share $\llbracket s \rrbracket_i = \hat{s}_i \bmod N$; otherwise $c'[i]$ is just uniformly random value in $\mathbb{Z}_{N^2}^*$ which decrypts to $(m[i] - m_1[i])r_i + \hat{s}_i \mod N$. Note that the value $(m[i] - m_1[i])r_i + \hat{s}_i$ is uniformly random over $\mathbb{Z}_N$ due to the fresh/random selection of $r_i \in \mathbb{Z}_N$.[6] Thus, if $m[i] \neq m_1[i]$, then the corresponding RRSS share $\llbracket s' \rrbracket_i = ((m[i] - m_1[i])r_i + \hat{s}_i \bmod N) \bmod 2^\lambda$ that we recover is randomly corrupted i.e., uniformly random over the range $[0, 2^\lambda)$. Thus, when $\mathtt{Ham}(m, m_1) \leq \ell$, we obtain $n - \ell$ correct shares and ℓ random ones, sufficient to recover K. Crucially, RRSS allows efficient identification of the valid shares, avoiding the $\binom{n}{\ell}$ time brute-force search required in [AB24]. When $\mathtt{Ham}(m, m_1) > \ell$, at least $\ell + 1$ shares are replaced with random values and the remaining $n - \ell - 1$ shares are indistinguishable from random by RRSS perfect privacy. Thus, all shares appear random. This enables straightforward simulation by sampling each $c'[i]$ uniformly from $\mathbb{Z}_{N^2}^*$.

The above description overlooks a few details e.g., if the RRSS share $\llbracket s \rrbracket_i$ might to too large for an individual Pallier ciphertext in which case we will need to split the share across several ciphertexts. We present the full

[6] Technically, to ensure that the value is uniformly random we require $\mathtt{gcd}(m[i] - m_1[i], N) = 1$. We can easily ensure that this property always holds by requiring that $\min\{p, q\} > |\Sigma|$ where $N = pq$ and $|\Sigma|$ is the size of the alphabet. Then we have $0 < m[i] - m_1[i] \leq |\Sigma| < \min\{p, q\}$.

construction in **Construction** 2. We prove security in Theorem 14 (details in Appendix H.3), along with correctness and standard real-or-random security (Appendix H.4, Appendix H.5).

6 Performance Evaluation and Implementation

In this section, we describe our C++ implementation of Random Robust Secret Sharing (RRSS) from **Construction** 1. We also present implementations of our fPAKE protocol and our conditional encryption scheme for arbitrary Hamming-distance predicates, both of which rely on RRSS. Our source code is available on an anonymous repository at `https://github.com/mhassanameri/RRSS`.

6.1 Implementation of Random Robust Secret Sharing

We implement our RRSS scheme as a modular, open-source C++ library designed for reuse in future applications and research prototypes. The implementation cleanly separates components for secret sharing, reconstruction, valid-share identification, and finite-field arithmetic.

Our valid-share identification procedure is based on Gauss–Jordan elimination. We implement this algorithm over $\mathbb{GF}(2^{\lambda_1})$, carefully handling finite-field inversion and row-reduction operations to ensure both correctness and efficiency. This functionality is encapsulated in a dedicated validation module and integrated as a wrapper component within the RRSS construction.

Finally, we implement a serialization and binding layer that converts RRSS inputs and outputs into byte streams, enabling the compiled C++ library to be invoked directly from Python. This facilitates integration with higher-level experimental frameworks and evaluation pipelines.

6.2 fPAKE using Random Robust Secret Sharing

Our fPAKE implementation extends the publicly available implementation of the (relaxed) fPAKE protocol by Fomichev et al. [FHA+21]. The original implementation relies on a robust secret sharing (RSS) primitive. We replace this component with our *Random Robust Secret Sharing* (RRSS) scheme while keeping the remaining protocol structure unchanged.

By integrating RRSS, we obtain full fPAKE security (i.e., robustness and $(t-1)$-perfect privacy simultaneously), thereby closing the gap present in prior practical implementations. To the best of our knowledge, this is the first practical implementation of an fPAKE protocol achieving full fPAKE security.

6.2.1 Experimental Methodology

We evaluate the computational and communication overhead introduced by replacing RSS with RRSS. Specifically, We compare RSS-based and RRSS-based implementations using: Computation time (local cryptographic processing), Communication time, Overall execution time, and Total communication overhead (in KiB). All other components of the protocol remain unchanged, allowing for a direct comparison between the original RSS-based construction and our RRSS-based variant.

Experiments were conducted using the same fingerprint inputs (derived from vehicle sensor data) and parameter settings as the original FastZip implementation. Fingerprints are represented as binary strings of length $n = 36$. We evaluate two sensor-derived input types and consider two security levels (128-bit and 244-bit), following the recommended parameter choices in [FHA+21].

Both protocol parties were executed locally to eliminate network variability and isolate protocol-induced overhead. We evaluated the performance our implementation of fPAKE using RRSS on a Ubuntu 24.04.3 LTS (Noble Numbat) with a 3.10 GHz 15-Core Intel® Core™ i9-9900 CPU processor and 64 GB 2666 MHz DDR4 RAM memory.

Algorithm	Sec.	Type	Role	Run time (calc)	Comm. time (net)	Overall time	Role comm. ovhd. (KiB)	Total comm. ovhd. (KiB)
fPAKE RSS	128	acc_h-bar	Sender	0.0123	0.0063	0.0186	3.3008	5.1289
			Receiver	0.0485	0.0119	0.0605	1.8281	5.1289
		acc_h-gyrW	Sender	0.0125	0.0065	0.0190	3.6523	5.6836
			Receiver	0.0626	0.0120	0.0746	2.0313	5.6836
	244	acc_h-bar	Sender	0.0351	0.0141	0.0492	5.0771	8.0303
			Receiver	0.0661	0.0289	0.0951	2.9531	8.0303
		acc_h-gyrW	Sender	0.0363	0.0155	0.0518	5.6162	8.8975
			Receiver	0.0854	0.0300	0.1153	3.2813	8.8975
fPAKE RRSS	128	acc_h-bar	Sender	0.2498	0.0083	0.2580	34.2031	36.0312
			Receiver	0.0447	0.2495	0.2943	1.8281	36.0312
		acc_h-gyrW	Sender	0.3387	0.0112	0.3499	41.7383	43.7695
			Receiver	0.0591	0.3385	0.3975	2.0313	43.7695
	244	acc_h-bar	Sender	0.2682	0.0166	0.2847	35.9795	38.9326
			Receiver	0.0588	0.2614	0.3202	2.9531	38.9326
		acc_h-gyrW	Sender	0.3547	0.0195	0.3742	43.7021	46.9834
			Receiver	0.0740	0.3466	0.4205	3.2813	46.9834

Table 1: fPAKE performance comparison across security levels, roles, and input types. Password length: $n = 36$

6.2.2 Implementation Adjustments

Beyond replacing RSS with RRSS, only minimal modifications were required. The first modification was related to Share Encoding. Since RRSS shares are larger than those used in the original implementation, we apply deterministic/pseudorandom key expansion to match share size prior to encoding. The second one was about handling communication. We adjusted handling of message to accommodate larger share sizes or communicated payloads. These modifications preserve the overall protocol structure, demonstrating that RRSS can be integrated into existing fPAKE deployments with minimal engineering effort.

6.2.3 Results

Table 1 compares the performance of our full-security fPAKE with RRSS with the relaxed-security fPAKE [FHA+21]. We report sender-side and receiver-side computation time, communication time, overall execution time, and communication overhead. We report performance across security levels $\lambda \in \{128, 244\}$, roles (sender/receiver), and fingerprint dataset (h-bar and h-gyrW) when the Hamming distance is at most $\ell_{\texttt{Ham}} = 3$.

Our results show that while RRSS increases communication overhead due to larger share sizes and strengthened robustness guarantees, the protocol remains practically deployable in vehicular communication settings.

Despite this overhead, execution times remain well below one second across all evaluated settings, demonstrating that full-security fPAKE with RRSS remains practically deployable. Importantly, this additional cost yields strictly stronger security guarantees: unlike prior practical implementations, our construction simultaneously achieves robustness against corrupted shares and $(t-1)$-perfect privacy, thereby eliminating the tradeoff between robustness and privacy inherent in earlier RSS-based designs. Thus, our results show that full fPAKE security can be obtained in practice with modest overhead.

In particular, empirical results from the FastZip [FHA+21] study indicate that deriving sufficient fingerprint entropy requires collecting ambient sensor data over a sliding window: approximately 10 seconds. In contrast, the computational and communication overhead introduced by our RRSS-based fPAKE protocol is well below one second, which is negligible compared to the fingerprint acquisition phase. Therefore, the additional cost of providing full fPAKE functionality does not significantly affect the overall execution time, as system latency is dominated by the sensor data collection process, demonstrating that RRSS remains practically deployable.

6.3 Conditional Encryption for Arbitrary Hamming Distance

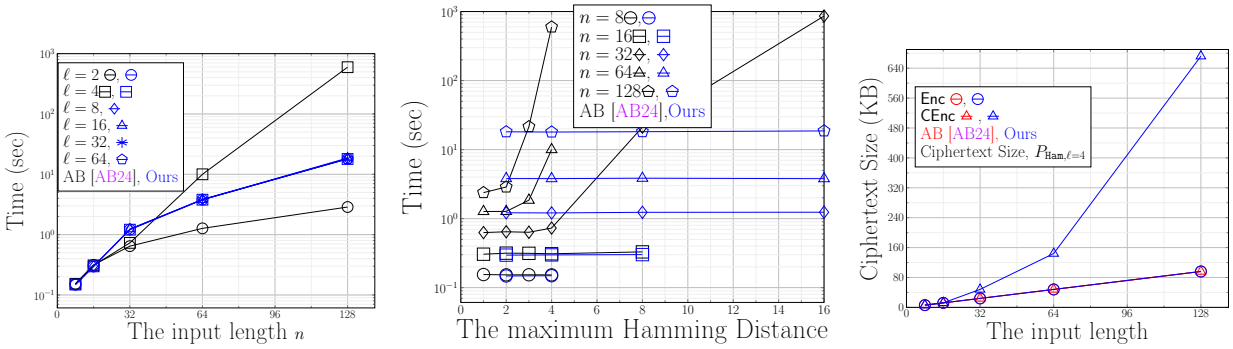
In this paper, we present an efficient conditional encryption scheme with a decryption algorithm which gives us an efficient conditional encryption for *arbitrary* Hamming distance. We already showed that the time complexity of the decryption algorithm is asymptotically reduced to $O(n-\ell)^2$ instead of $O\left(\binom{n}{\ell}\right)$.

6.3.1 Implementation

We implemented our conditional encryption schemes in C++. We implemented conditional encryption for a general Hamming distance predicate for an arbitrary value of ℓ (as the maximum Hamming distance). We defined our message space as $\Sigma^{\leq n}$ where Σ denotes the set of all ASCII characters and $n \in \{8, 16, 32, 64, 128\}$ — all $x \in \Sigma^{\leq n}$ are first padded to $\mathtt{Pad}(x) \in (\Sigma \cup \{y\})^n$ for a special new symbol y.

We used the Pallier Library [Cru16] as our implementation of the Pallier cryptosystem, and we used the GMP library [pro91] for computation with big integers. We instantiated Pallier with a 3072-bit modulus for 128-bit security [PDB+15].

Our implementation of conditional encryption schemes use CryptoPP library [Dai16] for Authenticated Encryption and Shamir Secret Sharing. For authenticated encryption, we use AES-GCM with 128 bit keys and use Shamir Secret Sharing over a field of size 2^{128} to generate the main shares of the secret symmetric key. CryptoPP implements Shamir secret sharing over the fixed field $\mathbb{GF}(2^{32})$. While this is suitable for sharing the 128-bit authenticated-encryption key, its 32-bit field elements are unnecessarily large for the additional Shamir instances in our RRSS construction. We therefore used the NTL library [Sho96] to implement Shamir secret sharing over the configurable field $\mathbb{GF}(2^{\lambda_1})$. We typically set $\lambda_1 = 8$, with two exceptions. When $n = 128$, we set $\lambda_1 = 9$ so that $2^{\lambda_1} > 2n$, ensuring sufficiently many distinct nonzero evaluation points. When $n = 8$, we set $\lambda_1 = 16$ to satisfy the RRSS requirement $\lambda_1 \geq \lceil \lambda/n \rceil = 128/8 = 16$.



(a) Conditional Decryption (b) Conditional Decryption (c) Reg/Cond Encryption

Figure 2: Performance Evaluation of our CE for Arbitrary Hamming Distance.

6.3.2 Evaluation

We evaluated the performance of our implementation of conditional encryption on the same machine used for the fPAKE application. We evaluate the performance of our conditional encryption schemes for the predicates $P_{\mathtt{Ham},\ell_1}$, and the results of these evaluations are presented in Figure 2. In particular, Figure 2b and Figure 2a evaluate the performance of our conditional encryption scheme for various Hamming distances $\ell = \{1, 2, 4, 8, 16, 32, 64\}$ and different padding lengths $n = \{8, 16, 32, 64, 128\}$. We highlight that for each padding length n we consider Hamming distances at most $\ell_n = \{i : i \in \ell, i < n\}$. For example, for $n = 128$, we have $\ell_{128} = \{1, 2, 4, 8, 16, 32, 64\}$ while for $n = 16$ we have $\ell_{16} = \{1, 2, 4, 8\}$. To show efficiency of our constructions, we compare the performance of our conditional encryption of arbitrary Hamming distance with Ameri and Blocki's conditional encryption (we use AB [AB24] to refer to their construction in Figure 2). The implementation of AB- [AB24] currently supports Hamming distances $\ell \in \{1, 2, 3, 4\}$ and various padding lengths $n \in \{8, 16, 32, 64, 128\}$, but was not feasible to run the conditional decryption algorithm for higher

values of ℓ when n is large, i.e., $n \in \{64, 128\}$ (however, we extend AB- [AB24] implementation to generate a few more plot points when $n = 32$ and $\ell \in \{8, 16\}$).

Discussion: Figure 2 shows that the construction of [AB24] is preferred in the special case of $\ell \leq 2$ since our construction has larger ciphertext size. However, the running time of their conditional decryption algorithm [AB24] quickly spikes as ℓ increases, while the running time for our construction is independent of ℓ — see Figure 2b. In situations where ℓ is larger, ours is the only practically viable option. For example, when $n = 32$ (a reasonable upper bound on the size of a user chosen password) and $\ell = 16$ the conditional decryption algorithm of [AB24] already takes about 14.35 minutes to complete and it was not even feasible to run the decryption algorithm for larger values of ℓ or n. By contrast, when $n = 32$ our conditional decryption algorithm runs in at most 1.2 seconds for any value of ℓ. As figure Figure 2c shows our construction does increase the size of of a conditional ciphertext though the size remains the same for regular ciphertexts. However, even when $n = 128$ the size of a conditional ciphertext remains manageable ($672KB$).

6.3.3 Comparison with Richardson et al. [RRX26]

In concurrent work, Richardson et al. [RRX26] construct an fPAKE protocol for arbitrary predicates using garbled circuits. An fPAKE protocol for the Hamming distance predicate can be obtained by instantiating either general-purpose construction ([RRX26] or [DHP+18]) with a circuit computing this predicate.

Asymptotic Comparison: A circuit computing the Hamming distance between two n-character passwords would have size $O(nk)$ yielding garbled circuits of size $O(nk\lambda)$ where $k = \log_2 |\Sigma|$ denotes the number of bits to encode each character from our alphabet Σ. By contrast, the RRSS shares in our fPAKE protocol have total size $O(n\lambda + n^2\lambda_1) = O(n\lambda + n^2 \log n)$. Thus, our protocol is asymptotically more efficient whenever $n \log n = o(k\lambda)$, whereas the protocol of Richardson et al. [RRX26] is asymptotically more efficient when $k\lambda = o(n \log n)$. We expect the former regime to be more relevant for user-chosen passwords i.e., $n \log n < k\lambda$. For example, with ASCII passwords ($k = 8$) and a $\lambda = 128$-bit security parameter, we have $n \log_2 n < k\lambda$ for every $n \leq 143$, well beyond the length of a typical user-chosen password.

A precise empirical comparison would require a full implementation of the protocols of Richardson et al. [RRX26] or Dupont et al. [DHP+18] using optimized circuits for the Hamming distance predicate. To our knowledge, no such implementation is currently available. Nevertheless, the asymptotic comparison above suggests that our protocol should compare favorably in the parameter regime relevant to user-chosen passwords. The garbled-circuit protocols must represent and process each password at the bit level, whereas our protocol operates directly on characters. Although our protocol becomes asymptotically less efficient as the password length n grows very large, password lengths are relatively small in practice. For example, more than 99.9% of the passwords in the RockYou dataset have length at most 30 [AB24].

7 Conclusion

We introduced the notion of random robust secret sharing (RRSS) with perfect privacy, resolving the inherent privacy–robustness gap in prior secret sharing schemes. We constructed RRSS and demonstrated its utility by designing a fuzzy password-authenticated key exchange (fPAKE) protocol with full UC security. To the best of our knowledge, this is the first practically efficient fPAKE achieving full UC security, and the first fully secure fPAKE protocol to be implemented; prior works [DHP+18, RRX25] rely on garbled circuits and lack implementations, while existing implementations [FHA+21] achieve weaker security due to this gap. We implement RRSS along with our fPAKE protocol and conditional encryption construction, demonstrating that these primitives are practical for realistic parameter settings. Beyond password authentication, our protocol may also enable zero-interaction device pairing based on noisy measurements [FHA+21].

We further showed how RRSS enables an efficient conditional encryption scheme for arbitrary Hamming distance, overcoming the limitations of prior constructions [AB24]. These results highlight RRSS as a versatile primitive for bridging theory and practice in cryptographic protocols over noisy secrets.

Acknowledgments.

The authors would like to thank the anonymous reviewers for their helpful feedback, which helped us improve the presentation of this paper. This research was supported in part by NSF CAREER Award CNS-2047272 and by an Amazon Research Award. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or Amazon.

References

[AB24] Mohammad Hassan Ameri and Jeremiah Blocki. Conditional encryption with applications to secure personalized password typo correction. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 4643–4657, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 361–377, Santa Barbara, CA, USA, August 14–18, 2005. Springer Berlin Heidelberg, Germany.

[BFH+23] Jonathan Bootle, Sebastian H. Faller, Julia Hesse, Kristina Hostáková, and Johannes Ottenhues. Generalized fuzzy password-authenticated key exchange from error correcting codes. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part VIII*, volume 14445 of *Lecture Notes in Computer Science*, pages 110–142, Guangzhou, China, December 4–8, 2023. Springer, Singapore, Singapore.

[BL23] Jeremiah Blocki and Peiyuan Liu. Towards a rigorous statistical analysis of empirical password datasets. In *2023 IEEE Symposium on Security and Privacy*, pages 606–625, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[Bon12] Joseph Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, San Francisco, CA, USA, May 21–23, 2012. IEEE Computer Society Press.

[BP16] Allison Bishop and Valerio Pastro. Robust secret sharing schemes against local adversaries. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 327–356, Taipei, Taiwan, March 6–9, 2016. Springer Berlin Heidelberg, Germany.

[BPRW16] Allison Bishop, Valerio Pastro, Rajmohan Rajaraman, and Daniel Wichs. Essentially optimal robust secret sharing with maximal corruptions. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 58–86, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.

[CAA+16] Rahul Chatterjee, Anish Athayle, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. pASSWORD tYPOS and how to correct them securely. In *2016 IEEE Symposium on Security and Privacy*, pages 799–818, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[CDD+15] Ronald Cramer, Ivan Bjerre Damgård, Nico Döttling, Serge Fehr, and Gabriele Spini. Linear secret sharing schemes from error correcting codes and universal hash functions. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 313–336, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany.

[CDF01] Ronald Cramer, Ivan Damgård, and Serge Fehr. On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 503–523, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.

[CDF+08] Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 471–488, Istanbul, Turkey, April 13–17, 2008. Springer Berlin Heidelberg, Germany.

[CDV94] Marco Carpentieri, Alfredo De Santis, and Ugo Vaccaro. Size of shares and probability of cheating in threshold schemes. In Tor Helleseth, editor, *Advances in Cryptology – EUROCRYPT'93*, volume 765 of *Lecture Notes in Computer Science*, pages 118–125, Lofthus, Norway, May 23–27, 1994. Springer Berlin Heidelberg, Germany.

[CFOR12] Alfonso Cevallos, Serge Fehr, Rafail Ostrovsky, and Yuval Rabani. Unconditionally-secure robust secret sharing with compact shares. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 195–208, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[Che19] Mahdi Cheraghchi. Nearly optimal robust secret sharing. *Designs, Codes and Cryptography*, 87(8):1777–1796, 2019.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 404–421, Aarhus, Denmark, May 22–26, 2005. Springer Berlin Heidelberg, Germany.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 597–608, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[Cru16] Mateus S. H. Cruz. Paillier cpp library, 2016. December, 31.

[CWP+17] Rahul Chatterjee, Joanne Woodage, Yuval Pnueli, Anusha Chowdhury, and Thomas Ristenpart. The TypTop system: Personalized typo-tolerant password checking. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 329–346, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[Dai16] Wei Dai. Cryptopp cpp library, 2016. Version 5.6.5, release date: 2016-10-11.

[DDWY90] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. In *31st Annual Symposium on Foundations of Computer Science*, pages 36–45, St. Louis, MO, USA, October 22–24, 1990. IEEE Computer Society Press.

[DHP+18] Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, and Sophia Yakoubov. Fuzzy password-authenticated key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 393–424, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

[FHA+21] Mikhail Fomichev, Julia Hesse, Lars Almon, Timm Lippert, Jun Han, and Matthias Hollick. Fastzip: faster and more secure zero-interaction pairing. In *Proceedings of the 19th Annual*

International Conference on Mobile Systems, Applications, and Services, MobiSys '21, page 440–452, New York, NY, USA, 2021. Association for Computing Machinery.

[FY19] Serge Fehr and Chen Yuan. Towards optimal robust secret sharing with security against a rushing adversary. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 472–499, Darmstadt, Germany, May 19–23, 2019. Springer, Cham, Switzerland.

[GGO10] Juan A. Garay, Clint Givens, and Rafail Ostrovsky. Secure message transmission with small public discussion. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 177–196, French Riviera, May 30 – June 3, 2010. Springer Berlin Heidelberg, Germany.

[GGOR14] Juan A. Garay, Clint Givens, Rafail Ostrovsky, and Pavel Raykov. Broadcast (and round) efficient verifiable secret sharing. In Carles Padró, editor, *ICITS 13: 7th International Conference on Information Theoretic Security*, volume 8317 of *Lecture Notes in Computer Science*, pages 200–219, Singapore, 2014. Springer, Cham, Switzerland.

[HO18] Brett Hemenway and Rafail Ostrovsky. Efficient robust secret sharing from expander graphs. *Cryptography and Communications*, 10(1):79–99, 2018.

[IOS12] Yuval Ishai, Rafail Ostrovsky, and Hakan Seyalioglu. Identifying cheaters without an honest majority. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 21–38, Taormina, Sicily, Italy, March 19–21, 2012. Springer Berlin Heidelberg, Germany.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.

[PDB+15] W Timothy Polk, Donna F Dodson, William E Burr, Hildegard Ferraiolo, and David Cooper. Cryptographic algorithms and key sizes for personal identity verification. *NIST Special Publication*, 800(78–4):1–19, 2015.

[pro91] GNU project. Gmp cpp library, 1991. Last update: 2023-07-30.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st Annual ACM Symposium on Theory of Computing*, pages 73–85, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[RRX25] David Richardson, Mike Rosulek, and Jiayu Xu. Conditionally input-revealing 2PC and fuzzy password-authenticated key exchange. Cryptology ePrint Archive, Report 2025/2315, 2025.

[RRX26] David Richardson, Mike Rosulek, and Jiayu Xu. Conditionally input-revealing 2PC and fuzzy password-authenticated key exchange. In Joan Daemen and Emmanuel Thomé, editors, *Advances in Cryptology – EUROCRYPT 2026, Part II*, volume 16542 of *Lecture Notes in Computer Science*, pages 468–498, Rome, Italy, May 10–14, 2026. Springer, Cham, Switzerland.

[RX23] Lawrence Roy and Jiayu Xu. A universally composable PAKE with zero communication cost - (and why it shouldn't be considered UC-secure). In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 714–743, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[Sho96] Victor Shoup. Ntl: A library for doing number theory, 1996. Version update: 2023-07-30.

[TW88] Martin Tompa and Heather Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, June 1988.

[WXXZ24] Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. New bounds for matrix multiplication: from alpha to omega. In David P. Woodruff, editor, *35th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 3792–3835, Alexandria, VA, USA, January 7–10, 2024. ACM-SIAM.

A Open Science

To support evaluation of the paper's core contributions, we provide a comprehensive open-source artifact. The artifact includes: (i) a C++ implementation of our RRSS scheme; (ii) the translator and supporting interface required to compile this implementation into a Python library for broader interoperability; (iii) a concrete implementation of our conditional encryption construction based on RRSS; and (iv) a Python re-implementation of the prior fPAKE realization in which the underlying robust secret sharing component is replaced with our RRSS, enabling end-to-end evaluation of the full construction studied in the paper. The artifact further contains the necessary build scripts, configuration files, dependency specifications, and usage documentation needed to reproduce the implementation results and experimental evaluation.

In particular, the implementation of our RRSS cleanly separates components for secret sharing, reconstruction, valid-share identification, and finite-field arithmetic.

For Shamir Secret Sharing over large finite fields, we rely on the CryptoPP library (https://github.com/weidai11/cryptopp) to generate the primary shares of the secret. In addition, we provide a lightweight implementation of Shamir Secret Sharing over the finite field $\mathbb{GF}(256)$ for performance-critical components of the protocol. Arithmetic over large integers and finite fields is supported via the NTL (https://libntl.org/) and GMP (https://gmplib.org/) libraries, which serve as dependencies of the CryptoPP backend.

The library is written in modern C++ with clear module boundaries and a reproducible build configuration using CMake. All experiments reported in this paper were compiled using standard optimization flags, and the implementation builds on commodity hardware without requiring specialized cryptographic accelerators. This design supports reproducibility and artifact evaluation.

Our implementation of conditional encryption schemes also use CryptoPP library for Authenticated Encryption and Shamir Secret Sharing. For authenticated encryption, we use AES-GCM with 128 bit keys and use Shamir Secret Sharing over a field of size 2^{128} to generate the main shares of the secret symmetric key.

Our implementation and experimental artifact are publicly available at https://github.com/mhassanameri/RRSS. The repository contains the full codebase corresponding to the contributions of the paper, and the accompanying README file describes the repository structure in detail, including the location and role of each component. It also provides complete instructions for setting up dependencies, compiling the implementations, and running the artifact.

B Ethical Considerations

Our experimental evaluation uses only publicly available, sanitized datasets released by prior work to support reproducible research. In particular, when comparing the empirical performance of our conditional encryption scheme with [AB24], we use password typo datasets from [CAA+16], which are publicly available at https://pages.cs.wisc.edu/~chatterjee/projects/pwtypos.html. These datasets were generated in the user studies of [CAA+16], where participants were asked to type randomly sampled passwords from the RockYou breach. Because these datasets are partially derived from breached-password data, we briefly discuss the ethical considerations of our use of them here.

The running time of our conditional decryption algorithm is independent of the underlying password and typo distribution, once password length is fixed. Thus, the performance of our conditional encryption scheme could in principle be evaluated using synthetic typo data. However, the conditional decryption algorithm of [AB24] depends significantly on the password typo distribution making it valuable to evaluate performance on a realistic password typo dataset. Accordingly, to enable a fair comparison and to maximize reproducibility, we use the same dataset as [AB24] in our evaluation.

To minimize risk, we do not directly use any breached dataset beyond the breached passwords that may already appear in the publicly released datasets of [CAA+16]. Moreover, we report only aggregate statistics from our performance analysis and do not publish any individual passwords. We therefore do not believe that our use of these datasets causes additional harm to users. At the same time, the fPAKE and conditional encryption protocols we develop are intended to improve user security and privacy.

We also compare the empirical performance of our fPAKE protocol with the implementation of the weaker fPAKE protocol from [FHA+21]. For this comparison, we use the same datasets as [FHA+21]. Fomichev et al. [FHA+21] collected these datasets from sensor measurements (accelerometer, gyroscope, and barometer) taken inside four different cars across a range of scenarios, including city roads, country roads, highways, and parking garages. These datasets were released publicly to support reproducibility. We do not believe that using these publicly available datasets raises significant ethical concerns, as they do not appear to contain private or confidential information.

C Comparison with Richardson et al. [RRX26]

In concurrent work, Richardson et al. [RRX26] construct an fPAKE protocol for arbitrary predicates using garbled circuits. Their protocol improves upon the prior construction of Dupont et al. [DHP+18] by reducing the round complexity from three to one and the number of garbled circuits from two to one. An fPAKE protocol for the Hamming distance predicate can be obtained by instantiating either general-purpose construction ([RRX26] or [DHP+18]) with a circuit computing this predicate.

C.1 Asymptotic Comparison

Let $k = \log_2 |\Sigma|$ denote the number of bits used to represent each character in our alphabet Σ. A circuit computing the Hamming distance between two n-character passwords would have size $O(nk)$ yielding garbled circuits of size $O(nk\lambda)$. By contrast, the RRSS shares in our fPAKE protocol have total size $O(n\lambda + n^2\lambda_1) = O(n\lambda + n^2 \log n)$. Thus, our protocol is asymptotically more efficient whenever $n \log n = o(k\lambda)$, whereas the protocol of Richardson et al. [RRX26] is asymptotically more efficient when $k\lambda = o(n \log n)$. We expect the former regime to be more relevant for user-chosen passwords i.e., $n \log n < k\lambda$. For example, with ASCII passwords ($k = 8$) and a $\lambda = 128$-bit security parameter, we have $n \log_2 n < k\lambda$ for every $n \leq 143$, well beyond the length of a typical user-chosen password.

C.2 Expected Concrete Performance

A precise empirical comparison would require a full implementation of the protocols of Richardson et al. [RRX26] or Dupont et al. [DHP+18] using optimized circuits for the Hamming distance predicate. To our knowledge, no such implementation is currently available. Nevertheless, the asymptotic comparison above suggests that our protocol should compare favorably in the parameter regime relevant to user-chosen passwords. The garbled-circuit protocols must represent and process each password at the bit level, whereas our protocol operates directly on characters. Although our protocol becomes asymptotically less efficient as the password length n grows, password lengths are relatively small in practice. For example, more than 99.9% of the passwords in the RockYou dataset have length at most 30 [AB24].

D Game Based Definitions for RRSS

In this section we provide an alternative definition for a Random Robust Secret Sharing (RRSS) scheme $\Pi_{\mathsf{RRSS}} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$ using game-based definitions.

Before giving the formal definition, it will be helpful to recall notation. Given a set $\mathbb{S}$, a vector $\vec{c} = (c_1, \ldots, c_n) \in \mathbb{S}^n$ of dimension n over $\mathbb{S}$ and a subset $S \subseteq [n]$ of indices, it will be useful to define $\mathcal{D}(\mathbb{S}, \vec{c}, S) \doteq \left\{\vec{c'} \in \mathbb{S}^n \ : \ \vec{c'}_S = \vec{c}_S\right\}$ as the set of all vectors $\vec{c'} = (c'_1, \ldots, c'_n) \in \mathbb{S}^n$ which are consistent with $\vec{c}$ over the indices in S i.e., such that $c'_i = c_i$ for all $i \in S$. Intuitively, if $\vec{c}$ denotes a vector of n shares then sampling $\vec{c'}$ uniformly at random from $\mathcal{D}(\mathbb{S}_\lambda, c, S)$ is equivalent to replacing c_i with a uniformly random sample $c'_i \in_R \mathbb{S}_\lambda$ for each $i \notin S$. t-random robustness tells us that if $|S| \geq t$ then (whp) we should still be able identify t uncorrupted shares and recover the secret.

Definition 2 *Let $\lambda \in \mathbb{N}$ be the security parameter, p a polynomial, $\mathbb{D}_\lambda = \{0,1\}^\lambda$ be the λ-bit secret space, $n, t \in \mathbb{N}$ with $t < n$, PP set of all public parameters and $\mathbb{S}_\lambda$ the space of all possible shares. An $(n, t, \epsilon(\lambda))$-Random Robust secret sharing scheme Π_{RRSS} contains four probabilistic polynomial time algorithms: setup $\mathsf{Setup} : 1^\lambda \times \mathbb{N} \times \mathbb{N} \to PP$, share generation $\mathsf{ShareGen} : \mathbb{D}_\lambda \times PP \to \mathbb{S}_\lambda^n$, reconstruction $\mathsf{SecretRecover} : 2^{[n]} \times \mathbb{S}_\lambda^t \times PP \to \mathbb{D}_\lambda$ and valid share identifier $\mathsf{Identify} : \mathbb{S}_\lambda^n \times PP \to 2^{[n]}$. Π_{RRSS} should satisfy three properties: correctness, $(t-1)$-perfect privacy and t-random robustness.*

- *Correctness Game $\mathtt{CORR}_{\mathcal{A},\lambda,n,t}(\Pi_{\mathsf{RRSS}})$: To begin the game the attacker $\mathcal{A}(\lambda, n, t)$ picks random coins r_{setup} and r_{share}, a secret $s \in \mathbb{D}_\lambda$ and a subset $S \subseteq [n]$. The $\mathcal{A}(\lambda, n, t)$ outputs the tuple $(r_{setup}, r_{share}, s, S)$. The challenger then deterministically computes $\mathsf{pp} := \mathsf{Setup}(1^\lambda, n, t; r_{setup})$ and $\vec{c} = (\llbracket c \rrbracket_1, \ldots, \llbracket c \rrbracket_n) := \mathsf{ShareGen}(s, \mathsf{pp}; r_{share})$. Finally, the challenger computes $s' = \mathsf{SecretRecover}(S, \vec{c}_S, \mathsf{pp})$. If $|S| \geq t$ and $s' \neq s$ then the attacker wins and the game outputs $\mathtt{CORR}_{\mathcal{A},\lambda,n,t}(\Pi_{\mathsf{RRSS}}) = 1$; Otherwise, the game outputs $\mathtt{CORR}_{\mathcal{A},\lambda,n,t}(\Pi_{\mathsf{RRSS}}) = 0$.*

  We say that Π_{RRSS} is correct if $\Pr[\mathtt{CORR}_{\mathcal{A},\lambda,n,t} = 1] = 0$ for all parameters λ, t, n with $t \leq n$ and every (unbounded) attacker $\mathcal{A}$ we have $s \in \mathbb{D}_\lambda$.

- *$(t-1)$-Perfect Privacy Game $\mathtt{PPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{\mathsf{RRSS}})$: To begin the game the attacker $\mathcal{A}(\lambda, n, t)$ picks random coins r_{setup}, a secret $s \in \mathbb{D}_\lambda$ and a subset $S \subseteq [n]$. In particular, $\mathcal{A}(\lambda, n, t)$ outputs the tuple $(r_{setup} s, S)$. The challenger then computes $\mathsf{pp} := \mathsf{Setup}(1^\lambda, n, t; r_{setup})$ and samples $\vec{c}^0 = (\llbracket c \rrbracket_1^0, \ldots, \llbracket c \rrbracket_n^0) \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$ using the honest $\mathsf{ShareGen}$ algorithm and samples $\vec{c}^1 = (\llbracket c \rrbracket_1^1, \ldots, \llbracket c \rrbracket_n^1) \leftarrow \mathbb{S}_\lambda^n$ uniformly at random (unrelated to s). Finally, the challenger samples a bit $b \in \{0,1\}$ and sends $\vec{c}_S^b$ to the attacker $\mathcal{A}$ who then outputs a guess $b' \in \{0,1\}$. If $b' = b$ and $|S| \leq t-1$ then the attacker wins the game and the output is $\mathtt{PPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1$; otherwise the game outputs $\mathtt{PPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 0$;*

  We say that Π_{RRSS} is perfectly private if for all (unbounded) attackers $\mathcal{A}$, and all parameters λ, t, n with $t \leq n$ we have $\Pr[\mathtt{PPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1] \leq \frac{1}{2}$.

- *t-Random Robustness Game $\mathtt{RROB}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS})$: To begin the game the attacker $\mathcal{A}(\lambda, n, t)$ picks random coins r_{setup} and r_{share}, a secret $s \in \mathbb{D}_\lambda$ and a subset $S \subseteq [n]$. The attacker $\mathcal{A}(\lambda, n, t)$ outputs the tuple $(r_{setup}, r_{share}, s, S)$. The challenger then deterministically computes $\mathsf{pp} := \mathsf{Setup}(1^\lambda, n, t; r_{setup})$ and $(\llbracket c \rrbracket_1, \ldots, \llbracket c \rrbracket_n) := \mathsf{ShareGen}(s, \mathsf{pp}; r_{share})$. The challenger then samples $\vec{c'} \leftarrow \mathcal{D}(\mathbb{S}_\lambda, c, S)$ i.e., so that $\vec{c'}_S = \vec{c}_S$ and $\vec{c'}$ is uniformly random on other coordinates. Finally, the challenger runs $S' = \mathsf{Identify}\left(\vec{c'}, \mathsf{pp}\right)$. The attacker wins if $|S| \geq t$, but $|S'| < t$ or $S' \not\subseteq S$. In this case the output of the game is $\mathtt{RROB}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1$. Otherwise, the output of the game is $\mathtt{RROB}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 0$.*

We say that the scheme Π_{RRSS} is t-random robust if for all for all (unbounded) attackers $\mathcal{A}$, and all parameters λ, t, n with $t \leq n$ we have $\Pr\left[\mathtt{RROB}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1\right] \leq \epsilon(\lambda)$.

Claim 7 *Suppose that Π_{RRSS} satisfies the original correctness property from Definition 1 then it satisfies the game based correctness property from Definition 2.*

Proof : Consider any (possibly unbounded) RRSS attacker $\mathcal{A}$ in the correctness game $\mathtt{CORR}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS})$. Consider any tuple $(r_{setup}, r_{share}, s, S)$ that $\mathcal{A}(\lambda, n, t)$ might output. WLOG assume that $|S| < t$ (otherwise $\mathcal{A}$ already lost). Let $\mathsf{pp} := \mathsf{Setup}(1^\lambda, n, t; r_{setup})$ and $(\llbracket c \rrbracket_1, \ldots, \llbracket c \rrbracket_n) := \mathsf{ShareGen}(s, \mathsf{pp}; r_{share})$ be the public parameters and shares generated by the challenger. Observe that, by definition pp is in the support of $\mathsf{Setup}(1^\lambda, n, t)$. Thus, by the non game based definition of correctness (Definition 2) we must have $\mathsf{SecretRecover}(S, \vec{c}_S, \mathsf{pp}) = s$ and the game will output $\mathtt{CORR}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 0$. Thus, for any RRSS attacker $\mathcal{A}$ we have $\Pr[\mathtt{CORR}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1] = 0$. □

Claim 8 *Suppose that Π_{RRSS} satisfies the original $(t-1)$-Perfect Privacy property Definition 1 then it satisfies the game based $(t-1)$-Perfect Privacy definition from Definition 2.*

Proof : Consider any (possibly unbounded) RRSS attacker $\mathcal{A}$ in the perfect privacy game $\mathtt{PPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS})$. Consider any tuple $(r_{setup}s, S)$ that $\mathcal{A}(\lambda, n, t)$ outputs at the beginning of the game. WLOG assume that $|S| < t$ (otherwise $\mathcal{A}$ already lost). Let $\mathsf{pp} := \mathsf{Setup}(1^\lambda, n, t; r_{setup})$ and $\vec{c}^0 = (\llbracket c \rrbracket_1^0, \ldots, \llbracket c \rrbracket_n^0) := \mathsf{ShareGen}(s, \mathsf{pp})$ be the public parameters and shares generated by the challenger (crucially, the randomness for the share generation algorithm $\mathsf{ShareGen}$ is not known to the attacker). Observe that, by definition pp is in the support of $\mathsf{Setup}(1^\lambda, n, t)$. Thus, by the non game based definition of perfect privacy (Definition 2) the distribution of $\vec{c}_S$ is uniformly random over $\mathbb{S}_\lambda^{|S|}$. Because $\vec{c}^1$ was picked uniformly at random over $\mathbb{S}_n^{|S|}$ the distribution of $\vec{c}_S^0$ and $\vec{c}_S^1$ are identical — both uniformly random over $\mathbb{S}_\lambda^{|S|}$. Since the distributions are perfectly indistinguishable we have $\Pr[b' = b] = \frac{1}{2}$. Thus, for any RRSS attacker $\mathcal{A}$ we have $\Pr[\mathtt{PPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1] \leq \frac{1}{2}$. □

Claim 9 *Suppose that Π_{RRSS} satisfies the original notion of (t)-Random Robustness as in Definition 1 then it satisfies the game based (t)-Random Robustness definition from Definition 2.*

Proof : Consider any (possibly unbounded) RRSS attacker $\mathcal{A}$ in the perfect privacy game $\mathtt{RROB}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS})$. Consider any tuple $(r_{setup}s, S)$ that $\mathcal{A}(\lambda, n, t)$ outputs at the beginning of the game. WLOG assume that $|S| \geq t$ (otherwise the attacker will already lose the game). Observe that, by definition, the public parameters $\mathsf{pp} = \mathsf{Setup}(1^\lambda, n, t; r_{setup})$ picked by the challenger is in the support of the Setup. Similarly, by definition, the shares $\vec{c} = (\llbracket c \rrbracket_1, \ldots, \llbracket c \rrbracket_n) := \mathsf{ShareGen}(s, \mathsf{pp}; r_{share})$ generated by the challenger are in the support of $\mathsf{ShareGen}(s, \mathsf{pp})$. Thus, by (t)-Random Robustness definition from Definition 2 when the challenger samples $\vec{c}' \leftarrow \mathcal{D}(\mathbb{S}_\lambda, c, S)$ we have $\Pr_{\vec{c}' \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}, S)}\left[S' = \mathsf{Identify}\left(\vec{c}', \mathsf{pp}\right) \subseteq S \wedge |S'| \geq t\right] \geq 1 - \epsilon(\lambda)$. If $S' = \mathsf{Identify}\left(\vec{c}', \mathsf{pp}\right) \subseteq S \wedge |S'| \geq t$ then, by definition, $\mathtt{RROB}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 0$ and the attacker loses. It follows that $\Pr\left[\mathtt{RROB}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1\right] \leq \epsilon(\lambda)$. □

D.0.1 Security Against Malicious Dealers

Game based definition: $\mathtt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS})$: To begin the game the attacker $\mathcal{A}(\lambda, n, t)$ picks random coins r_{setup}, and a subset $S \subseteq [n]$ and $\vec{c'} = (c'_1, \ldots, c'_n) \in \mathbb{S}_\lambda^n$. The attacker outputs $(r_{setup}, S, \vec{c})$. The challenger then deterministically computes $\mathsf{pp} := \mathsf{Setup}(1^\lambda, n, t; r_{setup})$. The challenger then samples $\vec{c''} \in_R \mathcal{D}\left(\mathbb{S}_\lambda, \vec{c'}, S\right)$. Intuitively, $c''_S = c'_S$ while c''_j is uniformly random for all $j \notin S$. If $|S| \geq t$ and $\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \neq \perp$ and $\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \not\subseteq S$ then the attacker wins and the game outputs $\mathtt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1$. Similarly, if $|S| < t$ and $\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \neq \perp$ then the attacker wins and the game outputs $\mathtt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1$. Otherwise, the game outputs $\mathtt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 0$.

Definition 3 *We say that the scheme Π_{RRSS} is $(t-1,\epsilon)$-secure against malicious dealers if for all for all (unbounded) attackers $\mathcal{A}$, and all parameters λ, t, n with $t \leq n$ we have* $\Pr\left[\texttt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1\right] \leq \epsilon(\lambda)$.

Claim 10 *Suppose that Π_{RRSS} is $(t-1,\epsilon(\lambda))$-secure against malicious dealers according to the original definition in Section 3.1.1. Then Π_{RRSS} is $(t-1,\epsilon(\lambda))$-secure against malicious dealers according to Definition 3.*

Proof : Consider any unbounded attacker $\mathcal{A}$ and let $(r_{setup}, S, \vec{c})$ denote the attacker's output at the beginning of the game. Let $\mathsf{pp} = \mathsf{Setup}(1^\lambda, n, t; r_{setup})$ denote the public parameters computed by the challenger and note that, by definition, pp is in the support of $\mathsf{Setup}(1^\lambda, n, t)$. We now consider two cases:
Case 1: $|S| \geq t$. In this case, the attacker only wins the game if $\mathsf{Identify}\left(\vec{c'}, \mathsf{pp}\right) \neq \perp$ and $\mathsf{Identify}\left(\vec{c'}, \mathsf{pp}\right) \not\subseteq S$ where c'' is selected uniformly at random from $\mathcal{D}\left(\mathbb{S}_\lambda, \vec{c'}, S\right)$ by the challenger. Thus, we have $\Pr\left[\texttt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1 \mid |S| \geq t\right] \leq \Pr\left[\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \neq \perp \wedge \mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \not\subseteq S\right]$ where the randomness of the second term is taken over the selection of $\vec{c''} \in_R \mathcal{D}\left(\mathbb{S}_\lambda, \vec{c'}, S\right)$

Because the original definition quantifies over all pp, c and $|S| \geq t$ we have

$$\Pr\left[\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \neq \perp \wedge \mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) \not\subseteq S\right] \leq \epsilon(\lambda)$$

where the randomness is taken over the selection of $\vec{c''} \in_R \mathcal{D}\left(\mathbb{S}_\lambda, \vec{c'}, S\right)$. It follows that $\Pr\left[\texttt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1 \mid |S| \geq t\right] \leq \epsilon(\lambda)$. **Case 2:** $|S| < t$. In this case the attacker wins simply if $\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right)$ where c'' is selected uniformly at random from $\mathcal{D}\left(\mathbb{S}_\lambda, \vec{c'}, S\right)$ by the challenger. Since the original definition quantifies over all $\vec{c'}, S$ and pp we have

$$\Pr_{\vec{c''} \in_R \mathcal{D}\left(\mathbb{S}_\lambda, \vec{c'}, S\right)}\left[\mathsf{Identify}\left(\vec{c''}, \mathsf{pp}\right) = \perp\right] \geq 1 - \epsilon(\lambda) \ .$$

It follows that $\Pr\left[\texttt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1 \mid |S| < t\right] \leq \epsilon(\lambda)$.

Finally, we have
$\Pr\left[\texttt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1\right] \leq \max\{p_1, p_2\} \leq \epsilon(\lambda)$ where $p_1 = \Pr\left[\texttt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1 \mid |S| \geq t\right]$ and $p_2 = \Pr\left[\texttt{MDPRIV}_{\mathcal{A},\lambda,n,t}(\Pi_{RRSS}) = 1 \mid |S| < t\right]$. □

E RRSS Security Proof

In this section, we provide the formal proofs of our RRSS Π_{RRSS}. While the protocol Π_{RRSS} was described completely in the main body, we describe it again here in the appendix (see **Construction** 1) in nicely formatted algorithmic environment for convenience.

E.1 Proof of Theorem 4 (Construction 1 is $(n, t, \epsilon(\lambda))$-RRSS)

In this section we prove that our **Construction** 1 construction is a Random Robust Secret Sharing Scheme which also achieves security against malicious delears — see Theorem 4. The proof of Theorem 4 relies on several useful claims about the system of equations defined in **Construction** 1. Claim 12 tells us that, except with negligible probability, any feasible solution $\vec{x} = (x_1, \ldots, x_{2n})$ to our system of linear equations must assign $x_{2j-1} = x_{2j} = 0$ for every corrupted share $j \notin S$. Similarly, Claim 13 tells us that, except with negligible probability, every feasible solution $\vec{x} = (x_1, \ldots, x_{2n})$ must have at least $|\{r \in [2n], x_r \neq 0\}| \geq 2t$ non-zero values. Thus, if a feasible solution exists it can be used to identify at least t valid shares with no

Construction 1

Setup: $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, n, t)$:

(1) Select $\lambda_1 \geq \lceil \frac{\lambda}{n} \rceil$, set $\Pi^{\lambda}_{SSS,n,t} = (\mathsf{ShareGen}_{2^\lambda}, \mathsf{SecretRecover}_{2^\lambda}), \Pi^{\lambda_1}_{SSS,2n,2t} = (\mathsf{ShareGen}_{2^{\lambda_1}}, \mathsf{SecretRecover}_{2^{\lambda_1}})$

(2) set $\vec{a} = (a_1, \ldots, a_{3n}) \in_R \mathbb{F}^{3n}_{2^{\lambda_1}} \setminus \{0\}$

(3) return $\mathsf{pp} = (\vec{a}, n, t, \Pi^{\lambda}_{SSS,n,t}, \Pi^{\lambda_1}_{SSS,2n,2t})$

Share Generation: $\left(\llbracket z \rrbracket_1, \ldots, \llbracket z \rrbracket_n\right) \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$:

(1) parse $\vec{a} = (a_1, \ldots, a_{3n}) \in \mathsf{pp}$.

(2) $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \Pi^{\lambda}_{SSS,n,t}.\mathsf{ShareGen}_{2^\lambda}(n, t, s)$

(3) $\forall j \in [3n]$: compute $\llbracket a \rrbracket^j_1, \ldots, \llbracket a \rrbracket^j_{2n} \leftarrow \Pi^{\lambda_1}_{SSS,2n,2t}.\mathsf{ShareGen}_{2^{\lambda_1}}(2n, 2t, a_j)$.

(4) $\forall i \in [n]$: Set $\llbracket z \rrbracket_i = \left(\llbracket s \rrbracket_i, \llbracket a \rrbracket^1_{2i-1}, \ldots, \llbracket a \rrbracket^{3n}_{2i-1}, \llbracket a \rrbracket^1_{2i}, \ldots, \llbracket a \rrbracket^{3n}_{2i}\right)$.

(5) return $\left(\left(\llbracket z \rrbracket_1 \ldots, \llbracket z \rrbracket_n\right)\right)$

Recovery from Valid Shares: $s := \mathsf{SecretRecover}(S, \vec{z}_S)$:

(1) parse $\vec{z}_S$ to recover $\left(\llbracket z \rrbracket_i\right)_{\forall i \in S}$

(2) $\forall i \in S$ extract $\llbracket s \rrbracket_i$ from $\llbracket z \rrbracket_i = \left(\llbracket s \rrbracket_i, \llbracket a \rrbracket^1_{2i-1}, \ldots, \llbracket a \rrbracket^{3n}_{2i-1}, \llbracket a \rrbracket^1_{2i}, \ldots, \llbracket a \rrbracket^{3n}_{2i}\right)$. Let $\vec{s}_S = \left(\llbracket s \rrbracket_i\right)_{i \in S}$.

(3) set $\vec{s}_S = \left(\llbracket s \rrbracket_i\right)_{i \in S}$

(4) return $s = \Pi^{\lambda}_{SSS,n,t}.\mathsf{SecretRecover}_{2^\lambda}(S, \vec{s}_S)$

Valid Share Identification: $S := \mathsf{Identify}\left(\left(\llbracket z \rrbracket_1 \ldots, \llbracket z \rrbracket_n\right), \mathsf{pp}\right)$ with $S \in 2^{[n]} \cup \{\perp\}$:

(1) parse $\vec{a} = (a_1, \ldots, a_{3n}) \in \mathsf{pp}$

(2) for all $i \in [n]$ parse: $\llbracket z \rrbracket_i = \left(s_i, \llbracket a \rrbracket^1_{2i-1}, \ldots, \llbracket a \rrbracket^{3n}_{2i-1}, \llbracket a \rrbracket^1_{2i}, \ldots, \llbracket a \rrbracket^{3n}_{2i}\right)$

(3) Form the system of linear equations $\vec{a} = A\vec{x}$ in which $A = \begin{bmatrix} a_{1,1} & \ldots & a_{1,2n} \\ \ldots & a_{i,j} & \ldots \\ a_{3n,1} & \ldots & a_{3n,2n} \end{bmatrix}$ where $a_{i,j} = \llbracket a \rrbracket^i_j$ and $\vec{x} = (x_1, \ldots, x_{2n})$ is a $2n$ unknown variable vector.

(4) Solve $\vec{a} = A\vec{x}$ to find the values $x_1, \ldots, x_{2n}$ and set $S_{\vec{x}} = \{i \in [n] : x_{2i-1} \neq 0 \vee x_{2i} \neq 0\}$. If there are no solutions then return $\perp$.

(5) If $|S_{\vec{x}}| < t$, then return $\perp$; otherwise, return $S_{\vec{x}}$ as the set of valid shares.

(6) return S.

false positives. Finally, Claim 11 implies that if at least $|S| \geq t$ shares are uncorrupted then there will be a feasible solution.

Reminder of Theorem 4. *The construction* $\Pi_{\mathsf{RRSS}} = (\mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$ *described in* ***Construction 1*** *is a* $(n, t, \epsilon(\lambda))$*-Random Robust Secret Sharing scheme with* $\epsilon(\lambda) \leq 2^{-\lambda}$. Π_{RRSS} *also achieves* $(t-1, \epsilon(\lambda))$*-Security against malicious dealers.*

Proof of Theorem 4: We first prove that our construction provides perfect $t-1$-perfect private, and then we show the construction provides t-random robustness and t-random robust share identification.

Proof of $(t-1)$-perfect Privacy. Fix any secret $s \in \mathbb{F}_{2^\lambda}$, any public parameters pp in the support of $\mathsf{Setup}(1^\lambda, n, t)$, and any subset $S \subseteq [n]$ with $|S| \leq t-1$. Consider sampling $((\llbracket z \rrbracket_1 \ldots, \llbracket z \rrbracket_n)) \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$. We show that the joint distribution of the shares $\vec{z}_S$ is uniform over $\mathbb{S}_\lambda$ where $\mathbb{S}_\lambda = \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^{\lambda_1}}^{6n}$. Recall that, after reordering, each RRSS share has the form $\llbracket z \rrbracket_i = \left(\llbracket s \rrbracket_i, \left(\llbracket a_j \rrbracket_{2i-1},\right)_{j \in [3n]}, \left(\llbracket a_j \rrbracket_{2i}\right)_{j \in [3n]},\right)$, where $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n)$ is a t-out-of-n Shamir sharing of s over $\mathbb{F}_{2^\lambda}$ and, independently for each $j \in [3n]$, $(\llbracket a_j \rrbracket_1, \ldots, \llbracket a_j \rrbracket_{2n})$ is a $2t$-out-of-$2n$ Shamir secret sharing of the public value $a_j \in \mathbb{F}_{2^{\lambda_1}} \setminus \{0\}$ — a_j is specified by pp.

By the perfect privacy of the t-out-of-n Shamir scheme (see Lemma 2), the vector $\left(\llbracket s \rrbracket_i\right)_{i \in S}$ is uniformly distributed over $\mathbb{F}_{2^\lambda}^{|S|}$, since $|S| \leq t-1$. Next define $T_S = \bigcup_{i \in S} \{2i-1, 2i\}$. We have $|T_S| \leq 2|S| \leq 2t-2 < 2t$. Consequently, for each fixed $j \in [3n]$, the perfect privacy of the $2t$-out-of-$2n$ Shamir scheme implies that $\left(\llbracket a_j \rrbracket_k\right)_{k \in T_S}$ is uniformly distributed over $\mathbb{F}_{2^{\lambda_1}}^{2|S|}$ even if we condition on the known public value a_j. Because, the Shamir Sharings of $s, a_1, \ldots, a_{3n}$ are generated independent the vectors above are all independent. It follows that $\vec{z}_S$ is uniformly distributed over $\left(\mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^{\lambda_1}}^{6n}\right)^{|S|} = \mathbb{S}_\lambda^{|S|}$.

***t*-Random Robustness.** In this part of the proof, we will show that our Π_{RRSS} construction provides t-Random Corrupted Share Identification which means that we can identify the valid shares with probability at least $1 - \epsilon(\lambda) = 1 - 2^{-n\lambda}$. Intuitively, we need to prove that if we have at least t valid shares in the given vector of n shares $((\llbracket z \rrbracket_1 \ldots, \llbracket z \rrbracket_n))$, our reconstruction algorithm will *correctly* identify the valid shares required for $\Pi^\lambda_{SSS,n,t}.\mathsf{SecretRecover}_{2^\lambda}$ to recover the main secret s. Once we identify the valid shares, we can recover the secret since our $\Pi^\lambda_{SSS,n,t}$ is an (n,t)-Shamir secret sharing and requires t valid shares to recover the secret s by executing $\Pi^\lambda_{SSS,n,t}.\mathsf{SecretRecover}_{2^\lambda}(\cdot)$ over the t valid shares as input. In fact, we can use these valid shares to reconstruct the coefficients of the underlying t-degree polynomial to recover the secret. We show that by forming and solving the system of linear equations we can identify the valid shares with high probability.

We begin by introducing some notation. Given any vector $\vec{c} = (\llbracket z \rrbracket_1, \ldots, \llbracket z \rrbracket_n) \in \mathbb{S}_\lambda^n$, let $A_{\vec{c}} \in \mathbb{F}_{2^{\lambda_1}}^{3n \times 2n}$ denote the matrix constructed in Step 3 of $\mathsf{Identify}(\vec{c}, \mathsf{pp})$. In particular, if $\llbracket z \rrbracket_i = (s_i, \llbracket a \rrbracket_{2i-1}^1, \ldots, \llbracket a \rrbracket_{2i-1}^{3n}, \llbracket a \rrbracket_{2i}^1, \ldots, \llbracket a \rrbracket_{2i}^{3n})$, then $A_{\vec{c}}[j,k] = \llbracket a \rrbracket_k^j$ for every $j \in [3n]$ and $k \in [2n]$. Thus, the system solved by $\mathsf{Identify}$ is $A_{\vec{c}}\vec{x} = \vec{a}$, where $\vec{x} \in \mathbb{F}_{2^{\lambda_1}}^{2n}$ and $\vec{a} = (a_1, \ldots, a_{3n}) \in \mathbb{F}_{2^{\lambda_1}}^{3n}$. For any assignment $\vec{x} = (x_1, \ldots, x_{2n})$, define $S_{\vec{x}} \doteq \{i \in [n] : x_{2i-1} \neq 0 \vee x_{2i} \neq 0\}$. If $\vec{x}$ is the solution selected in Step 4 of $\mathsf{Identify}$, then $S_{\vec{x}}$ is precisely the set of share indices identified by the algorithm.

Fix any public parameter pp in the support of $\mathsf{Setup}(1^\lambda, n, t)$, any secret $s \in \mathbb{F}_{2^\lambda}$, and any set $S \subseteq [n]$ with $|S| \geq t$. Sample an honest sharing $\vec{c} \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$ and then sample $\vec{c}' \leftarrow_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}, S)$. Thus, $\vec{c}'_S = \vec{c}_S$, while every share $\llbracket z' \rrbracket_i$ with $i \notin S$ is sampled independently and uniformly from $\mathbb{S}_\lambda$. We show that, except with probability at most $2^{-n\lambda_1}$, every solution to $A_{\vec{c}'}\vec{x} = \vec{a}$ identifies at least t shares, all of which belong to S.

We first observe that the system $A_{\vec{c}'}\vec{x} = \vec{a}$ always has a feasible solution $\vec{x}$ with $S_{\vec{x}} \subseteq S$. Choose any subset $T \subseteq S$ with $|T| = t$, and let $T^* := \{2i-1, 2i : i \in T\}$. For every $k \in T^*$, let x_k^* be the Lagrange

coefficient for evaluating at zero using the $2t$ Shamir evaluation points indexed by T^*, and set $x^*_k = 0$ for every $k \notin T^*$. Since each row of $A_{\vec{c}}$ consists of evaluations of a polynomial whose value at zero is the corresponding coordinate of $\vec{a}$, Lagrange interpolation gives $A_{\vec{c}}\vec{x}^* = \vec{a}$. Moreover, $S_{\vec{x}^*} = T \subseteq S$. Because $\vec{c}'_S = \vec{c}_S$, Claim 11 implies that $A_{\vec{c}'}\vec{x}^* = A_{\vec{c}}\vec{x}^* = \vec{a}$. Furthermore, observe that $S_{\vec{x}^*} = T \subseteq S$.

Call an assignment $\vec{x} \in \mathbb{F}^{2n}_{2^{\lambda_1}}$ *bad* if $A_{\vec{c}'}\vec{x} = \vec{a}$ and either $S_{\vec{x}} \not\subseteq S$ or $|S_{\vec{x}}| < t$. We bound the probability that a bad assignment exists. Fix an assignment $\vec{x}$ before sampling $\vec{c}$ and $\vec{c}'$. If $S_{\vec{x}} \not\subseteq S$, then Claim 12 implies that $\Pr[A_{\vec{c}'}\vec{x} = \vec{a}] \leq 2^{-3n\lambda_1}$.

Now suppose that $S_{\vec{x}} \subseteq S$ and $|S_{\vec{x}}| < t$. Since $\vec{c}'_S = \vec{c}_S$, Claim 11 implies that $A_{\vec{c}'}\vec{x} = A_{\vec{c}}\vec{x}$. Consequently, Claim 13 implies that $\Pr[A_{\vec{c}'}\vec{x} = \vec{a}] = \Pr[A_{\vec{c}}\vec{x} = \vec{a}] \leq 2^{-3n\lambda_1}$. Thus, for every fixed bad assignment $\vec{x}$, the probability that it satisfies the system is at most $2^{-3n\lambda_1}$.

There are $|\mathbb{F}^{2n}_{2^{\lambda_1}}| = 2^{2n\lambda_1}$ possible assignments $\vec{x}$. We can union bound over all $2^{2n\lambda_1}$ possible assignments $\vec{x}$ to upper bound the probability that there exists a bad assignment as $2^{-3n\lambda_1} \times 2^{2n\lambda_1} = 2^{-n\lambda_1} \leq 2^{-\lambda}$.

Thus, the linear system is feasible and, except with probability $2^{-\lambda}$, every feasible solution $\vec{x}$ satisfies $S_{\vec{x}} \subseteq S$ and $|S_{\vec{x}}| \geq t$. In this case it follows that $\mathsf{Identify}$ will output a set $S_{\vec{x}} \subseteq S$ of size $|S_{\vec{x}}| \geq t$. Therefore, $\Pr\left[S' = \mathsf{Identify}(\vec{c}', \mathsf{pp}) \subseteq S \wedge |S'| \geq t\right] \geq 1 - 2^{-n\lambda_1}$. Hence, the construction Π_{RRSS} satisfies t-random robustness with error $\epsilon(\lambda) = 2^{-n\lambda_1} \leq 2^{-\lambda}$.

.

Correctness. To prove that our construction provides this property, we need to show that if we are given $|S| \geq t$ valid shares, we can recover the secret s correctly with probability 1. Let S contain indexes of at least t valid shares (potentially can be obtained by running the $S = \mathsf{Identify}(\llbracket z \rrbracket_1, \ldots, \llbracket z \rrbracket_n, \mathsf{pp})$). Since the set S also identifies t valid shares Z_S of our Shamir secret sharing over the finite field of size 2^λ (Used to construct the original shares of the target secret s.). Referring to the perfect correctness of our Shamir secret sharing, we can reconstruct the secret s if and only if we have at least t valid shares.

Considering both properties t-random robustness and $(t-1)$-perfect privacy, we observe that **Construction 1** is a $\left(n, t, \left(\epsilon(\lambda_1) = 2^{-n\lambda_1}\right)\right)$-RRSS.

To finalize the proof of this theorem, we also need to show that **Construction 1** also provides $((t-1), \epsilon(\lambda_1))$-Security against malicious dealers. The proof of this part is discussed in Lemma 1. □

Notation: Let $\mathbb{S}_\lambda = \mathbb{F}_{2^\lambda} \times \mathbb{F}^{6n}_{2^{\lambda_1}}$ denote the set of all possible secret shares in our construction. Given any vector $\vec{c'} = (\llbracket z \rrbracket'_1, \ldots, \llbracket z \rrbracket'_n) \in \mathbb{S}^n_\lambda$ of n (possibly dishonest) shares let $A_{\vec{c}} \in \mathbb{F}^{3n \times 2n}_{2^{\lambda_1}}$ constructed in Step 3 of $\mathsf{Identify}(\vec{c'}, \mathsf{pp})$. In particular, if we parse each share as $\llbracket z \rrbracket_i = (s_i, \llbracket a \rrbracket^1_{2i-1}, \ldots, \llbracket a \rrbracket^{3n}_{2i-1}, \llbracket a \rrbracket^1_{2i}, \ldots, \llbracket a \rrbracket^{3n}_{2i})$ then $A_{\vec{c}}$ has coefficients $A_{\vec{c}}[j,k] = \llbracket a \rrbracket^j_k$ for every $j \in [3n]$ and $k \in [2n]$. Given an assignment $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}^{2n}_{2^{\lambda_1}}$ we let $S_{\vec{x}} \doteq \{i \in [n] : x_{2i-1} \neq 0 \ \vee \ x_{2i} \neq 0\}$.

If $\vec{x}$ is the solution found in Step 4 of $\mathsf{Identify}$ then $S_{\vec{x}}$ is precisely the set of share indices computed by the algorithm. It will be useful to prove a several helpful claims about the linear system. Suppose that $S \subseteq [n]$ (resp. $[n] \setminus S$) denotes the subset of shares that are correct (resp. randomly corrupted). Intuitively, Claim 12 tells us that, except with negligible probability, any feasible solution $\vec{x} = (x_1, \ldots, x_{2n})$ to our system of linear equations must assign $x_{2j-1} = x_{2j} = 0$ for every corrupted share $j \notin S$ so that $S_{\vec{x}} \subseteq S$. Similarly, Claim 13 tells us that, except with negligible probability, every feasible solution $\vec{x} = (x_1, \ldots, x_{2n})$ must have at least $|\{r \in [2n], x_r \neq 0\}| \geq 2t$ non-zero values. Thus, if a feasible solution exists it can be used to identify at least $|S_{\vec{x}}| \geq t$ valid shares with no false positives. Finally, Claim 11 implies that if at least $|S| \geq t$ shares are uncorrupted then there will be a feasible solution. To see this note that if $|S| \geq t$ then the original system of equations (without any share corruption) always has a feasible solution $\vec{x}$ with $S_{\vec{x}} \subseteq S$ obtained from the corresponding Lagrange coefficients in the Shamir Secret Sharing Scheme.

Claim 11 *For any set $S \subseteq [n]$, any $\vec{c}, \vec{c}' \in \mathbb{S}_\lambda$ with $\vec{c}_S = \vec{c}'_S$, and any $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}_{2^{\lambda_1}}^{2n}$ with $S_{\vec{x}} \subseteq S$ we have $A_{\vec{c}'}\vec{x} = A_{\vec{c}}\vec{x}$.*

Proof : Let $S \subseteq [n]$, $\vec{c}$, $\vec{c}'$ and $\vec{x}$ be given such that $\vec{c}_S = \vec{c}'_S$ and $S_{\vec{x}} \subseteq S$. We first claim that $x_j = 0$ for all $j \notin S^* \doteq \bigcup_{i \in S}\{2i-1, 2i\}$. Suppose for the sake of contradiction that $x_j \neq 0$. We can write $j = 2i - b$ for some $b \in \{0, 1\}$. Then we have $i \in S_{\vec{x}}$ by definition of $S_{\vec{x}}$. But this implies that $i \in S$, since $S_{\vec{x}} \subseteq S$, and that $j \in S^*$ by definition of S^*. Contradiction! Moreover, because $\vec{c}_S = \vec{c}'_S$ the matrices $A_{\vec{c}'}$ and $A_{\vec{c}}$ are, by definition, identical on all columns in the set S^*. Because $x_j = 0$ for all $j \notin S^*$ it follows that $A_{\vec{c}}\vec{x} = \sum_{k \in S^*} x_k A_{\vec{c}}[\cdot, k] = \sum_{k \in S^*} x_k A_{\vec{c}'}[\cdot, k] = A_{\vec{c}'}\vec{x}$. □

Claim 12 *For any set $S \subseteq [n]$, any $\vec{c}' \in \mathbb{S}_\lambda$ and any $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}_{2^{\lambda_1}}^{2n}$ with $S_{\vec{x}} \not\subseteq S$ and any vector $\vec{a} \in \mathbb{F}_{2^{\lambda_1}}^{n}$ we have*

$$\Pr_{\vec{c}'' \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}', S)} [A_{\vec{c}''}\vec{x} = \vec{a}] \leq 2^{-3n\lambda_1} \ .$$

Consequently,

$$\Pr_{\vec{c}'' \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}', S)} [\exists \vec{x}. S_{\vec{x}} \not\subseteq S \wedge A_{\vec{c}''}\vec{x} = \vec{a}] \leq 2^{-n\lambda_1} \leq 2^{-\lambda} \ .$$

Proof : Let $S \subseteq [n]$, $\vec{c}' \in \mathbb{S}_\lambda$, $\vec{a} \in (\mathbb{F}_{2^{\lambda_1}} \setminus \{0\})^n$ and $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}_{2^{\lambda_1}}^{2n}$ be given with $S_{\vec{x}} \not\subseteq S$.

Now consider randomly sampling $\vec{c''} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c'}, S)$ after $\vec{x}$ has been fixed. We can parse $\vec{c''} = (\llbracket z'' \rrbracket_1, \ldots, \llbracket z'' \rrbracket_n)$ and $\llbracket z'' \rrbracket_i = (s''_i, \llbracket a'' \rrbracket_{2i-1}^1, \ldots, \llbracket a'' \rrbracket_{2i-1}^{3n}, \llbracket a'' \rrbracket_{2i}^1, \ldots, \llbracket a'' \rrbracket_{2i}^{3n})$. Observe that for each $i \in [n] \setminus S$ the vector $\llbracket z'' \rrbracket_i = (s''_i, \llbracket a'' \rrbracket_{2i-1}^1, \ldots, \llbracket a'' \rrbracket_{2i-1}^{3n}, \llbracket a'' \rrbracket_{2i}^1, \ldots, \llbracket a'' \rrbracket_{2i}^{3n})$ is uniformly random over $\mathbb{S}_\lambda$. Now consider the system of $3n$ linear equations defined by $A_{\vec{c}''}\vec{x} = \vec{a}$ i.e., $\sum_{j=1}^{2n} \llbracket a'' \rrbracket_j^i x_j = a_i$ for each $i \leq 3n$.

Since $S_{\vec{x}} \not\subseteq S$ there is an index $j^\star \notin S$ such that $x_{2j^\star - 1} \neq 0$ or $x_{2j^\star} \neq 0$. Because $\llbracket c'' \rrbracket_{j^\star}$ is sampled uniformly from $\mathbb{S}_\lambda$, the values

$$\left(\llbracket a'' \rrbracket_{2j^\star - 1}^i, \llbracket a'' \rrbracket_{2j^\star}^i \right), \qquad i \in [3n],$$

are mutually independent and uniformly distributed over $\mathbb{F}_{2^{\lambda_1}}^2$. Fix any $i \in [3n]$ and WLOG suppose that $x_{2j^\star} \neq 0$ (the analysis when $x_{2j^\star - 1} \neq 0$ is identical). Because $x_{2j^\star} \neq 0$ the value $x_{2j^\star} \llbracket a'' \rrbracket_{2j^\star}^i \neq 0$ on the left side of the i'th equation is uniformly distributed over $\mathbb{F}_{2^{\lambda_1}}^2$. Thus, the i'th equation will be satisfied with probability exactly $2^{-\lambda_1}$. The relevant random field elements $x_{2j^\star} \llbracket a'' \rrbracket_{2j^\star}^1, \ldots, x_{2j^\star} \llbracket a'' \rrbracket_{2j^\star}^{3n}$ are independent across the $3n$ equations. Thus, the probability that all $3n$ equations are simultaneously satisfied by $\vec{x}$ is at most

$$\Pr_{\vec{c}'' \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}', S)} [A_{\vec{c}''}\vec{x} = \vec{a}] = 2^{-3n\lambda_1} \ .$$

The final claim follows by union bounding over all $2^{2n\lambda_1}$ possible assignments $\vec{x}$ and observing that $2^{-n\lambda_1} \leq 2^{-\lambda}$ because we chose λ_1 such that $n\lambda_1 \geq \lambda$. □

Claim 13 *For any public parameters $\mathsf{pp} \in \Pi_{\mathsf{RRSS}}.\mathsf{Setup}(1^\lambda, n, t)$ in the support of $\Pi_{\mathsf{RRSS}}.\mathsf{Setup}(1^\lambda, n, t)$, any secret $s \in \mathbb{F}_{2^\lambda}$ and any any $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}_{2^{\lambda_1}}^{2n}$ with $|S_{\vec{x}}| < t$ we have*

$$\Pr_{\vec{c} \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})} [A_{\vec{c}}\vec{x} = \vec{a}] \leq 2^{-3n\lambda_1} \ .$$

where $\vec{a} = (a_1, \ldots, a_{3n}) \in (\mathbb{F}_{2^{\lambda_1}} \setminus 0)^n$ is the vector defined by pp. Furthermore,

$$\Pr_{\vec{c} \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})} [\exists \vec{x}. |S_{\vec{x}}| < t \wedge A_{\vec{c}}\vec{x} = \vec{a}] \leq 2^{-n\lambda_1} \leq 2^{-\lambda} \ .$$

Proof : Let $\mathsf{pp} \in \Pi_{\mathsf{RRSS}}.\mathsf{Setup}(1^\lambda, n, t)$ be given and extract the vector $\vec{a} = (a_1, \ldots, a_{3n}) \in (\mathbb{F}_{2^{\lambda_1}})^n$ from pp. Note that, by the construction of $\Pi_{\mathsf{RRSS}}.\mathsf{Setup}$, we have $a_i \neq 0$ for all $i \leq 3n$. Let $s \in \mathbb{F}_{2^\lambda}$ and $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}_{2^{\lambda_1}}^{2n}$ be given with $|S_{\vec{x}}| < t$.

Consider sampling $\vec{c} \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$ after $\vec{x}$ is fixed and parse $\vec{c} = (\llbracket z \rrbracket_1, \ldots, \llbracket z \rrbracket_n)$ and $\llbracket z \rrbracket_i = (s_i, \llbracket a \rrbracket_{2i-1}^1, \ldots, \llbracket a \rrbracket_{2i-1}^{3n}, \llbracket a \rrbracket_{2i}^1, \ldots, \llbracket a \rrbracket_{2i}^{3n})$. We already proved that Π_{RRSS} satisfies $(t-1)$-perfect privacy (see the first section of the proof of Theorem 4). Thus, we can view the subvector $\vec{c}_{S_{\vec{x}}}$ as uniformly random. More specifically, for all $i \in S_{\vec{x}}$ and $j \leq 3n$ the values $\llbracket a \rrbracket_{2i-1}^j$ and $\llbracket a \rrbracket_{2i-1}^j$ are uniformly random over $\mathbb{F}_{2^{\lambda_1}}$ and independent.

Now consider the system of $3n$ linear equations defined by $A_{\vec{c}}\vec{x} = \vec{a}$ i.e., $\sum_{j=1}^{2n} \llbracket a \rrbracket_j^i x_j = a_i$ for each $i \leq 3n$. We can rewrite each the equation as $\sum_{j \in S_{\vec{x}}} \left(\llbracket a \rrbracket_{2j-1}^i x_{2j-1} + \llbracket a \rrbracket_{2j}^i x_{2j} \right) = a_i$.

As an edge case if $|S_{\vec{x}}| = 0$ is empty then we know that $\sum_{j \in S_{\vec{x}}} \left(\llbracket a \rrbracket_{2j-1}^i x_{2j-1} + \llbracket a \rrbracket_{2j}^i x_{2j} \right) = 0 \neq a_i$. Otherwise $|S_{\vec{x}}| > 1$ and the left side of the equation $\sum_{j \in S_{\vec{x}}} \left(\llbracket a \rrbracket_{2j-1}^i x_{2j-1} + \llbracket a \rrbracket_{2j}^i x_{2j} \right)$ is uniform over $\mathbb{F}_{2^{\lambda_1}}$. Thus, the probability that this particular equation is satisfied is at most $2^{-\lambda_1}$. By independence, the probability that all $3n$ equations are simultaneously satisfied is at most

$$\Pr_{\vec{c} \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})} [A_{\vec{c}}\vec{x} = \vec{a}] \leq 2^{-3n\lambda_1} .$$

The finally claim follows by union bounding over all $2^{2n\lambda_1}$ possible assignments $\vec{x}$ and observing that $2^{-n\lambda_1} \leq 2^{-\lambda}$ because we chose λ_1 such that $n\lambda_1 \geq \lambda$.

□

Lemma 1 ($(t-1, \epsilon(\lambda))$-Security against malicious dealers) *Let $\mathbb{S}_\lambda = \mathbb{F}_{2^\lambda} \times \mathbb{F}_{2^{\lambda_1}}^{6n}$ and let $\mathsf{pp} \in \Pi_{\mathsf{RRSS}}.\mathsf{Setup}(1^\lambda, n, t)$ be any valid public parameter in the support of $\Pi_{\mathsf{RRSS}}.\mathsf{Setup}(1^\lambda, n, t)$ then for any $S \subseteq [n]$ and any $\vec{c'} = (\llbracket z \rrbracket'_1, \ldots, \llbracket z \rrbracket'_n) \in \mathbb{S}_\lambda^n$ we have:*

$$\Pr_{\vec{c''} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c'}, S)} \left[\mathsf{Identify}(\vec{c''}, \mathsf{pp}) \neq \perp \wedge \mathsf{Identify}(\vec{c''}, \mathsf{pp}) \not\subseteq S \right] \leq 2^{-n\lambda_1} \leq 2^{-\lambda}$$

where the randomness is taken over the uniformly random choice of $\vec{c''}_i$ for all $i \notin S$. Similarly, for any $S \subseteq [n]$ with $|S| \leq t-1$ we have

$$\Pr_{\vec{c''} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c'}, S)} \left[\mathsf{Identify}(\vec{c''}, \mathsf{pp}) = \perp \right] \geq 1 - 2^{n\lambda_1} \geq 1 - 2^{-\lambda}$$

Proof of Lemma 1: Fix $\vec{c'} = (\llbracket z' \rrbracket_1, \ldots, \llbracket z' \rrbracket_n) \in \mathbb{S}_\lambda^n$ and $S \subseteq [n]$. Given our random sample $\vec{c''} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c'}, S)$ we can parse $\vec{c''} = (\llbracket z'' \rrbracket_1, \ldots, \llbracket z'' \rrbracket_n)$ and $\llbracket z'' \rrbracket_i = (s''_i, \llbracket a'' \rrbracket_{2i-1}^1, \ldots, \llbracket a'' \rrbracket_{2i-1}^{3n}, \llbracket a'' \rrbracket_{2i}^1, \ldots, \llbracket a'' \rrbracket_{2i}^{3n})$. Observe that for each $i \in [n] \setminus S$ the vector $\llbracket z'' \rrbracket_i = (s''_i, \llbracket a'' \rrbracket_{2i-1}^1, \ldots, \llbracket a'' \rrbracket_{2i-1}^{3n}, \llbracket a'' \rrbracket_{2i}^1, \ldots, \llbracket a'' \rrbracket_{2i}^{3n})$ is uniformly random over $\mathbb{S}_\lambda$.

Now consider the system of $3n$ linear equations constructed in Step 3 of $\mathsf{Identify}(\vec{c''}, \mathsf{pp})$ i.e., $\sum_{j=1}^{2n} \llbracket a'' \rrbracket_j^i x_j = a_i$ for each $i \leq 3n$ where $a_i \neq 0$ is fixed constant specified by the public parameters pp. We call an assignment $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}_{2^{\lambda_1}}^{2n}$ *misleading* for S if for some $i \notin S$ we have $x_{2i-1} \neq 0$ or $x_{2i} \neq 0$. In other words, $\vec{x}$ is misleading if the set $S_{\vec{x}} \doteq \{i \in [n] : x_{2i-1} \neq 0 \ \vee \ x_{2i} \neq 0\}$ associated with the assignment $\vec{x}$ is not contained in S. By Claim 11, except with probability $2^{-n\lambda_1} \leq 2^{-\lambda}$, no misleading assignment $\vec{x}$ exists which satisfies $A_{\vec{c''}}\vec{x} = \vec{a}$.

Observe that $\mathsf{Identify}$ will either return $\perp$ or $S_{\vec{x}}$ for some satisfying assignment $\vec{x}$. If no misleading system satisfies the system of equations, then it immediately follows that if the identification algorithm finds a feasible solution $\vec{x}$ in step 4 that $\vec{x}$ is not misleading for S. Thus, we must have $S_{\vec{x}} \subseteq S$ for the set $S_{\vec{x}}$ constructed in step 4. This proves that $\Pr_{\vec{c'} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}, S)} \left[\mathsf{Identify}(\vec{c'}, \mathsf{pp}) \neq \perp \wedge \mathsf{Identify}(\vec{c'}, \mathsf{pp}) \not\subseteq S\right] \leq 2^{-n\lambda_1} \leq 2^{-\lambda}$.

Finally, suppose that $|S| \leq t-1$. We claim that if there is no misleading assignment that satisfies all $3n$ equations then $\mathsf{Identify}$ will always output $\perp$. To see this observe that, by construction, $\mathsf{Identify}$ will only return $S_{\vec{x}}$ if $\vec{x}$ is a satisfying assignment and $|S_{\vec{x}}| \geq t$ — if $|S_{\vec{x}}| < t$ then $\mathsf{Identify}$ will return $\perp$. However, if $|S_{\vec{x}}| \geq t$ and $|S| \leq t-1$ then $S_{\vec{x}} \not\subseteq S$ which implies that $\vec{x}$ is misleading. It follows that $\Pr_{\vec{c'} \in_R \mathcal{D}(\mathbb{S}_\lambda, \vec{c}, S)} \left[\mathsf{Identify}(\vec{c'}, \mathsf{pp}) = \perp\right] \geq 1 - 2^{n\lambda_1} \geq 1 - 2^{\lambda}$.

□

Let $\vec{c} = (\llbracket z \rrbracket_1, \ldots, \llbracket z \rrbracket_n) \leftarrow \mathsf{ShareGen}(s, \mathsf{pp})$ denote the honest shares for a secret s. We can parse each $\llbracket z \rrbracket_i = (s_i, \llbracket a \rrbracket^1_{2i-1}, \ldots, \llbracket a \rrbracket^{3n}_{2i-1}, \llbracket a \rrbracket^1_{2i}, \ldots, \llbracket a \rrbracket^{3n}_{2i})$ and define the matrix $A_{\vec{c}}$ with coefficients $A_{\vec{c}}[i,j] = \llbracket a \rrbracket^i_j$. We say that a variable assignment $\vec{x} = (x_1, \ldots, x_{2n}) \in \mathbb{F}^{2n}_{2^{\lambda_1}}$ is *too small* if $|S_{\vec{x}}| < t$ where $S_{\vec{x}} \doteq \{i \in [n] : x_{2i-1} \neq 0 \ \vee \ x_{2i} \neq 0\}$. For any s, pp and any assignment $\vec{x}$ with $|S_{\vec{x}}| < t$ we have $\Pr_{\vec{c} \leftarrow \mathsf{ShareGen}(s,\mathsf{pp})}[(\vec{a})^T = A\vec{x}^T] \leq 2^{-3n\lambda_1}$ where $a_{i,j} = \llbracket a \rrbracket^i_j$ and the randomness is taken over the random coins of the share generation algorithm.

F Missing Proofs of fPAKE from RRSS

F.1 fPAKE Correctness Definition and Proof of Theorem F.1

Definition 4 (fPAKE correctness [BFH+23]) *We say* fPAKE *protocol* π *is* (ℓ, ϵ)*-correct if for all* $\mathtt{pwd}, \mathtt{pwd}' \in \mathcal{PWD}$ *with* $\mathtt{Ham}(\mathtt{pwd}, \mathtt{pwd}') \leq \ell$*, where* $\mathcal{PWD}$ *is the set of all passwords, and for all passive adversaries* $\mathcal{A}$ *(honest but curious) the probability that an execution of* π *with honest players* Party_0 *and* Party_1 *with input passwords* $\mathtt{pwd}$ *and* $\mathtt{pwd}'$*, respectively output different keys in the presence of* $\mathcal{A}$ *is bounded as:*

$$\Pr[K \neq K' | (K, K') \leftarrow out_{\mathcal{A},\pi}\langle \mathsf{Party}_0(\mathtt{pwd}), \mathsf{Party}_1(\mathtt{pwd}')\rangle] \leq \epsilon(\lambda).$$

where $(K, K') \leftarrow out_{\mathcal{A},\pi}\langle \mathsf{Party}_0(\mathtt{pwd}), \mathsf{Party}_1(\mathtt{pwd}')\rangle$ *denotes outputting* K *and* K' *to* Party_0 *and* Party_1 *respectively by executing the protocol* π*, and the probability is takes over the random coin of* $\mathsf{Party}_0, \mathsf{Party}_1$*, the adversary* $\mathcal{A}$ *and the protocol* π*.*

Reminder of Theorem F.1. *(*fPAKE *Correctness) If* Π_{RRSS} *is as an* $(n, n-\ell, \epsilon_{\mathsf{RRSS}}(\lambda))$*-random robust secret sharing,* π *is* $\epsilon_{\mathsf{iPAKE}}(\lambda)$*-correct, the hash function* $\mathcal{H}$ *UC-realizes* $\mathcal{F}_{\mathsf{GRO}}$ *and UC-realizes* $\mathcal{F}_{\mathsf{iPAKE}}$*, then the protocol in Figure 1 is* (ℓ, ϵ)*-correct for* $\epsilon(\lambda) = n\epsilon_{\mathsf{iPAKE}}(\lambda) + \epsilon_{\mathsf{RRSS}}(\lambda)$ *where* q *upper bounds the total number of queries to* $\mathcal{H}$*.*

Proof of Theorem F.1: Assume that Party_0 and Party_1 execute the protocol in Figure 1 honestly on inputs $\mathtt{pwd}_0$ and $\mathtt{pwd}_1$ with $\mathtt{Ham}(\mathtt{pwd}_0, \mathtt{pwd}_1) \leq \ell$, and let $\mathcal{A}$ be a passive adversary so that sid and $\mathsf{sid}_{\mathbb{H}}$ are consistent for each party. Thus, as long as $s = s'$ in the final step the final output keys $K = \mathcal{H}((\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}), s)$ and $K' = \mathcal{H}((\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}), s')$ will be identical.

By assumption, each execution of the underlying iPAKE protocol π_j is $\epsilon_{\mathsf{iPAKE}}(\lambda)$-correct (Definition 5).

We define a sequence of $n+1$ hybrids $\mathcal{H}_0, \ldots, \mathcal{H}_n$ as follows: in hybrid $\mathcal{H}_j$, we abort if the j-th instance π_j outputs different keys despite being run on matching inputs $\mathtt{pwd}_j^{\mathsf{Party}_0} = \mathtt{pwd}_j^{\mathsf{Party}_1}$. (If the inputs differ, we do not abort.)

By $\epsilon_{\mathsf{iPAKE}}(\lambda)$-correctness of π, each consecutive pair $\mathcal{H}_{j-1}$ and $\mathcal{H}_j$ differs with probability at most $\epsilon_{\mathsf{iPAKE}}(\lambda)$. By a union bound over all n hybrids, the probability that any matching-character instance produces inconsistent keys is at most $n\epsilon_{\mathsf{iPAKE}}(\lambda)$. In the final hybrid $\mathcal{H}_n$ we are guaranteed that, if we don't abort, that we have $K_j = K'_j$ for at least $n - \ell$ keys since there are at least $n - \ell$ indices j for which $\mathtt{pwd}_j^{\mathsf{Party}_0} = \mathtt{pwd}_j^{\mathsf{Party}_1}$. For indices $j \in [n]$ where $\mathtt{pwd}_j^{\mathsf{Party}_0} \neq \mathtt{pwd}_j^{\mathsf{Party}_1}$ the key K'_j is uniformly random and independent of K_j.

Consequently, the reconstructed shares $\mathbf{C}' = \mathbf{E} \oplus \mathbf{K}'$ contain at least $n - \ell$ valid shares (equivalently, at most ℓ shares are randomly corrupted) and every invalid share was randomly corrupted. Since Π_{RRSS} is an $(n, n-\ell, \epsilon_{\mathsf{RRSS}}(\lambda))$-RRSS scheme, the secret s is recovered with probability at least $1 - \epsilon_{\mathsf{RRSS}}(\lambda)$ whenever at least $n - \ell$ shares are correct.

Combining all of the error events yields

$$\begin{aligned}&\Pr\left[K \neq K' \middle| (K, K') \leftarrow out_{\mathcal{A},\pi}\left\langle \mathsf{Party}_0\left(\mathtt{pwd}^{\mathsf{Party}_0}\right), \mathsf{Party}_1\left(\mathtt{pwd}^{\mathsf{Party}_1}\right)\right\rangle\right] \\ &\quad \leq n\epsilon_{\mathsf{iPAKE}}(\lambda) + \epsilon_{\mathsf{RRSS}}(\lambda)\end{aligned}$$

□

F.2 Proof of Theorem 6

Before presenting the security proof, we describe the ideal functionality for fPAKE. We include its full specification in Figure 3, as it is needed for the proof.

The functionality interacts with Party_0, Party_1, and the adversary $\mathcal{A}$ via three queries: NewSession, TestPwd, and NewKey.

Our functionality closely follows Dupont et al. [DHP+18], with one key difference. Prior work introduced a larger leakage threshold $\gamma \geq \ell$, allowing information leakage whenever $\mathtt{Ham}(\mathtt{pwd}, \mathtt{pwd}') \leq \gamma$. We eliminate this relaxation: a password guess either satisfies $\mathtt{Ham}(\mathtt{pwd}, \mathtt{pwd}') \leq \ell$ (in which case key exchange succeeds), or it is treated as an incorrect guess with no additional leakage. Thus, our functionality is strictly stronger than those in [DHP+18, BFH+23].

Execution proceeds as follows. A party initiates a session via NewSession, submitting a session identifier, role (sender or receiver), and password pwd. The functionality records the session and informs $\mathcal{A}$ without revealing pwd. The adversary may issue a single TestPwd query per fresh session to attempt a password guess. If $\mathtt{Ham}(\mathtt{pwd}, \mathtt{pwd}') \leq \ell$, the record is marked compromised and $\mathcal{A}$ learns the matching character positions; otherwise, the session is marked interrupted and no further information is revealed.

If the adversary guesses a sufficiently *close* password (Hamming distance at most ℓ), then the record associated with pwd is marked compromised. Otherwise the record is marked as interrupted. The ideal functionality informs the adversary about the correctness of their guess. The ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$ is parametrized by error tolerance ℓ and the Hamming distance as the associated metric and we formally say that two password pwd and pwd′ are close if $d = \mathtt{Ham}(\mathtt{pwd}, \mathtt{pwd}') \leq \ell$. So if $d \leq \ell$, the ideal functionality also leaks the location of the matching characters of the tested password pwd′. In [BFH+23], the ideal functionality also leaks the locations of the matching characters in the password to the adversary when the password guess is incorrect but satisfies $\ell < d \leq \gamma$ for some parameter γ which is related to the robust secret sharing scheme that they use. By contrast, we do not weaken the ideal functionality to allow for leakage when $\ell < d \leq \gamma$. Thus, our ideal functionality is strictly stronger than $\mathcal{F}_{\mathsf{fPAKE}}$ in [DHP+18, BFH+23].

Upon a NewKey query, the functionality determines the output key according to three cases:

(i) If $\mathcal{A}$ correctly guessed a password (via TestPwd or corruption), it specifies the resulting key.

(ii) If both parties are honest and $\mathtt{Ham}(\mathtt{pwd}_0, \mathtt{pwd}_1) \leq \ell$, they receive the same key.

(iii) In all other cases, each party receives a/an different/independent uniformly random key.

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

The functionality $\mathcal{F}_{\mathsf{fPAKE}}$ is parametrized by the security parameter λ and the Hamming distance tolerance ℓ. The ideal functionality interacts with the two players Party_0 and Party_1 and the adversary $\mathcal{A}$ via the following queries:

New Session: On input $(\textsc{NewSession}, \mathsf{sid}, \mathtt{pwd}_i, \mathsf{role}])$ from partyParty_i, where $\mathtt{pwd}_i$ is a password and $\mathsf{role} = \mathsf{sender}$ implies Party_i wishes to initiate a key exchange, while $\mathsf{role} = \mathsf{receiver}$ implies Party_i wishes to respond:

- Send $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathsf{role})$ to $\mathcal{A}$
- If one of the following is true, record $(\mathsf{Party}_i, \mathtt{pwd}_i)$ and mark this record `fresh`.
  - This is the first NewSession query;
  - This the second NewSession query and there is a record $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$.

Test Password: On input $(\textsc{TestPwd}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}^*)$ from the adversary $\mathcal{A}$: if there is a `fresh` record $(\mathsf{Party}_i, \mathtt{pwd}_i)$, then set $d \leftarrow \mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}^*)$ and do:

- If $d \leq \ell$, mark the record `compromised` and reply $\mathcal{A}$ with $(\textsc{CorrectGuess}, L_c(\mathtt{pwd}_i, \mathtt{pwd}^*))$ in which $L_c(\mathtt{pwd}_i, \mathtt{pwd}^*) = \{j \text{ s.t. } \mathtt{pwd}_i[j] = \mathtt{pwd}^*[j]\}$.
- If $d > \ell$, mark the record `interrupted` and reply to $\mathcal{A}$ with $(\textsc{WrongGuess})$.

New Key: On $(\textsc{NewKey}, \mathsf{sid}, \mathsf{Party}_i, K)$ from the adversary $\mathcal{A}$, if there is not record of the form $(\mathsf{Party}_i, \mathtt{pwd}_i)$, or if this is not the first NewKey query for Party_i, then ignore this query. Otherwise:

- If at least one of the following is true, then output (sid, K) to player Party_i:
  - The record is `compromised`;
  - The record is `fresh`, Party_{1-i} is corrupted, and there is a record $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$ with $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}_{1-i}) \leq \ell$.
- If this record is `fresh`, both parties are honest, there is a record $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$ with $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}_{1-i}) \leq \ell$, a key K' was sent to Party_{1-i}, and $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$ was fresh at the time, then output (sid, K') to Party_i.
- In any other case, pick a new random key K' of length λ and send (sid, K') to Party_i.
- Mark the record $(\mathsf{Party}_i, \mathtt{pwd}_i)$ as `completed`.

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$ for fPAKE [DHP+18].

This captures the core guarantee of fPAKE: parties with sufficiently close passwords obtain the same key, while otherwise learning nothing beyond failure.

Reminder of Theorem 6. *If* Π_{RRSS} *is an* $(n, n-\ell, \epsilon_{\mathsf{RRSS}}(\lambda))$*-random robust secret sharing which additionally provides* $(n-\ell-1, \epsilon_{\mathsf{RRSS}}(\lambda))$*-security against malicious dealers, the hash function* $\mathcal{H}$ *UC-realizes* $\mathcal{F}_{\mathsf{GRO}}$*, and* π *is* $(t_{\mathsf{iPAKE}}(\lambda), \epsilon_{\mathsf{iPAKE}}(\lambda))$*-universally compossible which realizes* $\mathcal{F}_{\mathsf{iPAKE}}$ *using simulator* $\mathtt{Sim}_{\mathsf{iPAKE}}$ *in presence of the environment* $\mathcal{Z}^*$ *running in time at most* $t_{\mathsf{iPAKE}}(\lambda)$*, then the protocol in Figure 1 UC-realizes* $\mathcal{F}_{\mathsf{fPAKE}}$ *in the* $\mathcal{F}_{\mathsf{SA}}$*-hybrid model with error tolerance in password* ℓ *and leakage threshold* $\gamma = \ell$*.*

Proof of Theorem 6: To prove this theorem, we need to show that no PPT environment $\mathcal{Z}$ has non-negligible advantage in distinguishing a real execution of the protocol in Figure 1 from interaction with the ideal world abstraction $\mathcal{F}_{\mathsf{fPAKE}}$ and the simulator described in Figure 4 and Figure 5.

So, we define a series of hybrids $\mathbf{H_0}$ to $\mathbf{H_{11}}$ similar to what we have in proof of Theorem 2 in [BFH+23]. We focus on the differences and highlight them.

- $\mathbf{H_0}$: This is the real execution of the protocol in Figure 1.
- $\mathbf{H_1}$: **Create functionality and simulator**. This hybrid is exactly the same as $\mathbf{G_1}$ in [BFH+23]. More specifically, in this game we introduce a simulator and reformulate the experiment so that the execution of the real protocol is carried out inside the simulator. We also introduce dummy parties and an ideal functionality that simply forwards messages to enable this execution. When a party initiates a session with input $(\textsc{NewSession}, \mathsf{sid}, \mathtt{pwd}_i, \mathsf{role})$, the dummy party sends this input to the ideal functionality , which forwards $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}_i, \mathsf{role})$ to the simulator without modifying the password. The simulator then internally runs the protocol instances using the parties' real passwords and produces the corresponding session keys. The functionality additionally provides an interface $(\textsc{NewKey}, \mathsf{sid}, \mathsf{Party}_i, sk)$ that allows the simulator to instruct the functionality to deliver a chosen key sk as the output of party Party_i. Since the protocol execution and outputs remain exactly the same as in the real experiment—only the location of execution changes—this transformation is purely syntactic, and the distributions in the real experiment and H_0 and H_1 are information theoretically identical and we have
$$\Pr[H_1] = \Pr[H_0]$$
.
- $\mathbf{H_2}$: **Abort on Collision of** $\mathcal{H}$. This hybrid is also the same as the $\mathbf{G_2}$ in [BFH+23]. More specifically, we modify the simulator so that it aborts the execution if a collision occurs in the underlying hash function $\mathcal{H}$ which UC-Realizes our $\mathcal{F}_{\mathsf{GRO}}$. More precisely, the simulator terminates the simulation whenever there exist two distinct inputs s and s' with $s \neq s'$ such that $\mathcal{H}((\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}), s) = \mathcal{H}((\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}), s')$. Since the outputs of $\mathcal{H}$ are modeled as being chosen uniformly at random for each new query, the probability that a collision occurs among polynomially many queries is negligible. Therefore, the probability that the simulator aborts in this game is negligibly small, and the distributions in the previous hybrid and this game remain computationally indistinguishable and we have:
$$|\Pr[H_2] - \Pr[H_1]| \leq \mathsf{negl}(\lambda) \ .$$
- $\mathbf{H_3}$: **Exchanging real** iPAKE **for simulated ones**. This hybrid is also the same as the $\mathbf{G_3}$ in [BFH+23]. We introduce a binary variable MitM indicating whether the environment $\mathcal{Z}$ executes a man-in-the-middle attack. In particular, upon receiving an initialization message $(\textsc{Init}, \mathsf{sid}, \mathsf{Party}_i, \mathbb{H}, \mathsf{sid}_{\mathbb{H}})$ from the environment $\mathcal{Z}$ on behalf of the simulated functionality $\mathcal{F}_{\mathsf{SA}}$, the simulator $\mathtt{Sim}$ sets $\mathsf{MitM} = 0$ if $\mathbb{H} = \{\mathsf{Party}_0, \mathsf{Party}_1\}$, namely, if the session contains exactly the two intended parties, and sets $\mathsf{MitM} = 1$, otherwise.

  For every instance of π_{iPAKE}, the simulator $\mathtt{Sim}$ runs an internal instance of $\mathcal{F}_{\mathsf{iPAKE}}$ together with a corresponding simulator $\mathtt{Sim}_{\mathsf{iPAKE}}$. More precisely, whenever Party_i starts the j-th instance of

π_{iPAKE} with session identifier $(\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}, j)$ and password character $\mathtt{pwd}_i[j]$, the simulator creates an instance $\mathcal{F}^j_{\mathsf{iPAKE}}$ with the same session identifier and provides, on behalf of Party_i, the input $(\text{NEWSESSION}, (\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}, j), \mathtt{pwd}_i[j], \mathsf{Party}_i)$. It also invokes a corresponding simulator $\mathtt{Sim}^j_{\mathsf{iPAKE}}$, whose existence follows from the assumption that π_{iPAKE} UC-realizes $\mathcal{F}_{\mathsf{iPAKE}}$. Since $\mathtt{Sim}$ still knows the parties' passwords in this hybrid, it can provide each instance of $\mathcal{F}_{\mathsf{iPAKE}}$ with the corresponding password character.

When $\mathcal{F}^j_{\mathsf{iPAKE}}$ instructs $\mathtt{Sim}^j_{\mathsf{iPAKE}}$ to deliver a NEWKEY output to Party_i, the simulator $\mathtt{Sim}$ delivers that output to Party_i as if it had been generated by the corresponding real instance of π_{iPAKE} protocol. Moreover, after all n simulators $\mathtt{Sim}^1_{\mathsf{iPAKE}}, \ldots, \mathtt{Sim}^n_{\mathsf{iPAKE}}$ associated with Party_i have produced their outgoing messages $\vec{Z}^* = (Z_1^*, \ldots, Z_n^*)$, the simulator sends $\vec{Z}^*$ to $\mathcal{F}_{\mathsf{SA}}$ on behalf of Party_i. Conversely, whenever $\mathcal{F}_{\mathsf{SA}}$ delivers a vector $(Z_1^*, \ldots, Z_n^*)$ to Party_i, the simulator instructs each $\mathtt{Sim}^j_{\mathsf{iPAKE}}$ to process Z_j^* as the message delivered by the environment to its corresponding instance. If $\mathsf{MitM} = 0$, the parties use the same $\mathsf{sid}_{\mathbb{H}}$, and there are n paired instances of $\mathcal{F}_{\mathsf{iPAKE}}$. If $\mathsf{MitM} = 1$, the two parties may use different subsession identifiers and may therefore induce up to $2n$ instances, with at most n instances associated with each party. We replace these real π_{iPAKE} instances one at a time. By the UC security of π_{iPAKE}, replacing any one instance with the corresponding $\mathtt{Sim}^j_{\mathsf{iPAKE}}/\mathcal{F}^j_{\mathsf{iPAKE}}$ execution changes the environment's output (the environment who want distinguish between these gradual changes) by probability at most $\epsilon_{\mathsf{iPAKE}}(\lambda)$. Since at most $2n$ such replacements are required, a hybrid argument and the union bound give the following distinguishing advantage:

$$|\Pr[\mathbf{H_3}] - \Pr[\mathbf{H_2}]| \leq 2n\epsilon_{\mathsf{iPAKE}}(\lambda)$$

- $\mathbf{H_4}$: **Functionality makes records of running sessions, marks them, and reacts accordingly.** This hybrid is also the same as the $\mathbf{G_4}$ in [BFH+23]. In this hybrid, we modify only the ideal functionality. Upon receiving $(\text{NEWSESSION}, \mathsf{sid}, \mathtt{pwd}_i, \mathsf{role})$ from a dummy Party_i, the functionality records $\langle \mathsf{Party}_i, \mathsf{sid}, \mathsf{role}, \mathtt{pwd}_i \rangle$ and marks this record as `fresh` if either this is the first NEWSESSION query for session sid, or this is the second such query and a corresponding record $\langle \mathsf{Party}_{1-i}, \mathsf{sid}, \mathsf{role}', \mathtt{pwd}_{1-i} \rangle$ already exists. Thus, the functionality now implements the full NEWSESSION interface of $\mathcal{F}_{\mathsf{fPAKE}}$, except that it still forwards the parties' passwords to $\mathtt{Sim}$.

  We next equip the functionality with the TESTPWD interface of $\mathcal{F}_{\mathsf{fPAKE}}$. Consequently, a fresh record may be marked `compromised` following a successful password test or `interrupted` following an unsuccessful password test. We also modify the behavior of the functionality upon receiving $(\text{NEWKEY}, \mathsf{sid}, \mathsf{Party}_i, k)$ from $\mathtt{Sim}$. The functionality first checks whether a record $\langle \mathsf{Party}_i, \mathsf{sid}, \mathsf{role}, \mathtt{pwd}_i \rangle$ exists and whether this is the first NEWKEY query for Party_i. If these conditions are not satisfied, it ignores the query. Otherwise, it processes the query according to the status of the record. If the record is marked `interrupted`, the functionality chooses a fresh uniformly random key and delivers it to Party_i. If the record is marked `compromised`, or the applicable corrupted-party condition in the NEWKEY interface of $\mathcal{F}_{\mathsf{fPAKE}}$ holds, it delivers the key k supplied by $\mathtt{Sim}$. Importantly, when the record is `fresh`, the functionality still delivers the key provided by $\mathtt{Sim}$, as in the preceding hybrid; this behavior will be changed in a later hybrids. After processing the query, the functionality marks the record as `completed`.

  The newly introduced records and markings are internal to the functionality and can affect the externally visible execution only through the NEWKEY interface. At this point in the hybrid sequence, however, $\mathtt{Sim}$ never invokes the TESTPWD interface of $\mathcal{F}_{\mathsf{fPAKE}}$. Therefore, no record can be marked `interrupted`. Moreover, in all remaining cases, the functionality continues to deliver the key supplied by $\mathtt{Sim}$, exactly as in $\mathbf{H_3}$. Hence, these changes are purely syntactic, and the output distributions of the two hybrids are identical. Thus,

$$\Pr[\mathbf{H_3}] = \Pr[\mathbf{H_4}]$$

- **$\mathbf{H_5}$: Simulator handles TestPwd queries of $\mathcal{F}_{\mathsf{fPAKE}}$ for honest and undisturbed sessions**. This hybrid is also the same as the $\mathbf{G_5}$ in [BFH+23]. In the following, we provide a brief summary of the high-level intuition and the necessary details of each hybrid. In this hybrid, we modify only the simulator's behavior for an honest session, namely, a session in which both parties are honest and $\mathsf{MitM} = 0$. Whenever $\mathtt{Sim}$ receives a TestPwd query from an internal simulator $\mathtt{Sim}^j_{\mathsf{iPAKE}}$ for such a session, it ignores the query instead of using the parties' passwords to answer it. Consequently, for honest sessions, $\mathtt{Sim}$ can simulate the NewSession and TestPwd interfaces of the internal $\mathcal{F}_{\mathsf{iPAKE}}$ instances without using the parties' passwords. At this point, however, $\mathtt{Sim}$ still uses the parties' passwords to simulate the reaction of the internal $\mathcal{F}_{\mathsf{iPAKE}}$ instances to NewKey queries. Thus, this hybrid constitutes the first step toward eliminating the simulator's dependence on the honest parties' passwords.

  It remains to argue that ignoring these TestPwd queries changes the distribution only with negligible probability. When both parties are honest and $\mathsf{MitM} = 0$, they receive the same authentication-set identifier, i.e., $\mathsf{sid}_{\mathbb{H}} = \mathsf{sid}_{\mathbb{H}'}$, from $\mathcal{F}_{\mathsf{SA}}$. Hence, the adversary cannot modify or insert messages into the corresponding authenticated iPAKE executions: it can either deliver the messages unchanged or prevent the sessions from completing.

  In an honest and undisturbed iPAKE execution, the simulator $\mathtt{Sim}^j_{\mathsf{iPAKE}}$ does not know the password character and therefore cannot issue an effective TestPwd query except with negligible probability. Indeed, an incorrect password test would mark the corresponding internal $\mathcal{F}_{\mathsf{iPAKE}}$ record as interrupted and could cause the two parties to receive independent keys even when their password characters match. This would contradict the correctness of π_{iPAKE} for an honest execution. Since π_{iPAKE} is correct and UC-realizes $\mathcal{F}_{\mathsf{iPAKE}}$, the probability that any $\mathtt{Sim}^j_{\mathsf{iPAKE}}$ issues such a query in an honest and undisturbed session is negligible. Therefore, ignoring these queries changes the output distribution by at most a negligible amount, and we have:

$$|\Pr[H_4] - \Pr[H_5]| \leq \mathsf{negl}(\lambda).$$

- **$\mathbf{H_6}$: Simulator handles attacks on honest senders**. This hybrid is the same as $\mathbf{G_6}$ in [BFH+23], except that we use the security properties of Π_{RRSS} in place of the query-uniform error-correcting code used there.

  In this hybrid, we modify the simulator's behavior when the adversary attacks an honest sender Party_i. For every position $j \in [n]$, let c_j denote the password character appearing in the TestPwd query issued by the corresponding internal simulator $\mathtt{Sim}^j_{\mathsf{iPAKE}}$. The simulator combines these characters into the candidate password $\mathtt{pwd}^* := c_1 \| c_2 \| \cdots \| c_n$ and submits $(\text{TestPwd}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}^*)$ to $\mathcal{F}_{\mathsf{fPAKE}}$. Let $J := \{j \in [n] : c_j = \mathtt{pwd}_i[j]\}$ and $t := n - \ell$. Thus, $\mathtt{Ham}(\mathtt{pwd}_i, \mathtt{pwd}^*) \leq \ell$ if and only if $|J| \geq t$. If the password test succeeds, $\mathcal{F}_{\mathsf{fPAKE}}$ marks the sender's record as $\mathtt{compromised}$ and reveals the set J of matching positions to $\mathtt{Sim}$. The simulator then uses this set to answer the individual TestPwd queries of the internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instances. If the password test fails, $\mathcal{F}_{\mathsf{fPAKE}}$ marks the record as $\mathtt{interrupted}$ and does not reveal J. In this intermediate hybrid, $\mathtt{Sim}$ still uses the honest sender's password to determine J and answer the individual TestPwd queries in this unsuccessful case. This remaining use of the password will be removed in a subsequent hybrid.

  We next explain why the resulting distribution can be simulated using $\mathcal{F}_{\mathsf{fPAKE}}$. Let E_j be the masked RRSS share transmitted at position j. If $j \in J$, the two corresponding password characters match, and the internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instance provides the same session key used by the honest sender. Consequently, unmasking E_j yields the genuine RRSS share s_j. If $j \notin J$, the unsuccessful password test causes the corresponding internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instance to provide an independently uniform session key K_j. Hence, for every $x \in \mathcal{X}_{\mathsf{RRSS}}$, $\Pr[E_j \oplus K_j = x \mid E_j] = \frac{1}{|\mathcal{X}_{\mathsf{RRSS}}|}$, where $\mathcal{X}_{\mathsf{RRSS}}$ denotes the RRSS share domain. Thus, every matching position yields a genuine share, whereas every nonmatching position yields an independently uniform corrupted share.

(1) **At least $t = n - \ell$ characters are guessed correctly.** In this case, $|J| \geq t$, and hence there are at most $n - t = \ell$ independently uniform corrupted shares. By the robustness of Π_{RRSS} against uniformly random corruption pattern, reconstruction returns the sender's secret s_{send} except with probability at most $\epsilon_{\mathsf{RRSS}}(\lambda)$. Conditioned on successful reconstruction, the malicious receiver obtains the same session key as in $\mathbf{H_5}$. The fact that $\mathcal{F}_{\mathsf{fPAKE}}$ now marks the corresponding sender record as `compromised` allows `Sim` to provide precisely this key. Therefore, the executions in this case differ only if RRSS reconstruction fails.

(2) **Fewer than $t = n - \ell$ characters are guessed correctly.** In this case, $|J| \leq t - 1$. As described above, the receiver's reconstructed share vector contains at most $t - 1$ genuine RRSS shares, while every remaining component is an independently uniform element of $\mathcal{X}_{\mathsf{RRSS}}$. By the $(t-1)$-perfect privacy of Π_{RRSS}, the genuine shares indexed by J are statistically independent of the shared secret s_{send}. Because the remaining components are also independently uniform, the receiver's entire reconstructed-share vector is independent of s_{send}. It follows that the adversary cannot distinguish the honest sender's key $\mathcal{H}(\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}, s_{\mathsf{send}})$ from a fresh uniformly random key unless it previously queried the random oracle at $(\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}, s_{\mathsf{send}})$. Since s_{send} is uniform and hidden from the adversary, an adversary making at most $q_{\mathcal{H}}(\lambda)$ random-oracle queries causes this event with probability at most $q_{\mathcal{H}}(\lambda)/|\mathcal{X}_{\mathsf{RRSS}}|$, which is negligible. Therefore, the sender's key in $\mathbf{H_5}$ is indistinguishable from the fresh uniformly random key produced by $\mathcal{F}_{\mathsf{fPAKE}}$ for the `interrupted` record in $\mathbf{H_6}$.

Combining the two cases, we have:

$$|\Pr[\mathbf{H_6}] - \Pr[\mathbf{H_5}]| \leq \epsilon_{\mathsf{RRSS}}(\lambda) + \frac{q_{\mathcal{H}}(\lambda)}{|\mathcal{X}_{\mathsf{RRSS}}|} \leq \mathsf{negl}(\lambda).$$

- $\mathbf{H_7}$: **Use random iPAKE keys in case of attack on honest sender**: As in $\mathbf{H_6}$, we modify the simulator only when a man-in-the-middle attack is mounted against an honest sender or when the receiver is corrupted. Let $J := \{j \in [n] : c_j = \mathtt{pwd}_{\mathsf{sender}}[j]\}$ and $t := n - \ell$. If $\mathcal{Z}$'s password guess is correct, i.e., $|J| \geq t$, the simulator behaves exactly as in $\mathbf{H_6}$, and therefore the distribution is unchanged (since the guess is correct, the ideal functionality reveal J to `Sim`). Suppose instead that $|J| \leq t - 1$ and $\mathcal{Z}$'s guess is correct on at most $t - 1$ characters. In $\mathbf{H_6}$, the simulator uses the honest sender's password to identify the positions in J (since the guess is wrong and the ideal functionality does not reveal J). Consequently, the value recovered at position j is $\widetilde{s}_j = \begin{cases} s_j, & j \in J, \\ E_j \oplus K_j, & j \notin J, \end{cases}$ where s_j is the genuine RRSS share and, for every $j \notin J$, K_j is an independently uniform iPAKE key. Thus, the values $\{\widetilde{s}_j\}_{j \notin J}$ are independently uniform elements of the RRSS share domain. In this hybrid, the simulator no longer uses the honest sender's password to identify the positions in J. Instead, it uses independently uniform iPAKE keys for all internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instances associated with the sender. Hence, all coordinates of the resulting reconstructed-share vector are independently uniform. It remains to compare these two distributions. Since $|J| \leq t-1$, the $(t-1)$-perfect privacy of Π_{RRSS} guarantees that the share vector $(s_j)_{j \in J}$ in $\mathbf{H_6}$ is jointly uniform over its share domain. Moreover, it is independent of the uniformly masked coordinates indexed by $[n] \setminus J$. Therefore, the entire reconstructed share vector in $\mathbf{H_6}$ is already distributed as a vector of independently uniform elements, exactly as in $\mathbf{H_7}$. Because the internal iPAKE outputs are not revealed separately to $\mathcal{Z}$, replacing all of them with random keys does not change the adversary's (or $\mathcal{Z}$'s) view. Thus, in the case $|J| \leq t - 1$, the two hybrids are information theoretically identical. Combining the two cases, we have:

$$\Pr[\mathbf{H_7}] = \Pr[\mathbf{H_6}].$$

We emphasize that, in this hybrid, `Sim` no longer uses the honest sender's password when handling a corrupted receiver or a man-in-the-middle attack. If the candidate password satisfies $|J| \geq t = n - \ell$,

then $\mathcal{F}_{\mathsf{fPAKE}}$ marks the sender's record as `compromised` and reveals the set J of matching positions to `Sim`. The simulator uses only this leaked set, rather than the sender's password, to answer the individual TestPwd queries of the internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instances. If $|J| \leq t-1$, then $\mathcal{F}_{\mathsf{fPAKE}}$ marks the record as `interrupted` and does not reveal the matching-position set. In this case, `Sim` does not attempt to determine which individual character guesses were correct. Instead, it uses independently uniform iPAKE keys for **all** corresponding internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instances. The $(t-1)$-perfect privacy of Π_{RRSS} guarantees that this produces exactly the same distribution as in $\mathbf{H_6}$. Thus, in both cases, attacks on an honest sender are simulated without using the sender's password.

- $\mathbf{H_8}$: **Simulator handle attacks on honest receivers.** In this hybrid, we modify the simulator's behavior when a man-in-the-middle attack is mounted against an honest receiver, or when the sender is corrupted. This hybrid corresponds to $\mathbf{G_8}$ in [BFH+23].

  Let $Q \subseteq [n]$ denote the set of positions j for which the internal simulator $\mathtt{Sim}^j_{\mathsf{iPAKE}}$ issued a TestPwd query, and let c_j be the character contained in that query. Define $J := \{j \in Q : c_j = \mathsf{pwd}_{\mathsf{receiver}}[j]\}$ and $t := n - \ell$. The set J is introduced only for the analysis; the simulator does not compute it using the honest receiver's password. After receiving the NewKey queries from all internal $\mathtt{Sim}^j_{\mathsf{iPAKE}}$ instances, the simulator proceeds as follows. If $|Q| \geq t$, it completes the recorded character guesses to a candidate password $\mathtt{pwd}^*$ by independently sampling characters at the positions outside Q, and submits $(\text{TestPwd}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}^*)$ to $\mathcal{F}_{\mathsf{fPAKE}}$. If $|Q| < t$, the simulator submits a password-test query that is guaranteed to fail, thereby marking the receiver's record as `interrupted`. We distinguish the following two cases.

  Case 1: The password test succeeds. In this case, $\mathtt{pwd}^*$ is within Hamming distance ℓ of the honest receiver's password. The functionality marks the receiver's record as `compromised` and reveals the matching-position set to `Sim`. The simulator uses this leakage to determine the outputs of the corresponding internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instances; it does not use the receiver's password. The simulator can then run the honest receiver's RRSS reconstruction algorithm on the resulting simulated transcript. Because the record is marked `compromised`, the NewKey interface allows `Sim` to specify the receiver's output. The simulator therefore submits the same key that the receiver would have computed in $\mathbf{H_7}$. Consequently, the distribution in this case is identical to that in the preceding hybrid.

  Case 2: The password test fails, or fewer than t individual password-test queries were issued. In either situation, at most $t-1$ internal iPAKE instances can have provided keys corresponding to correct character guesses. More precisely, $|J| \leq t-1$. For every position $j \in J$, the value recovered from the malicious sender's masked value E_j may be an arbitrary RRSS share selected by that (malicious) sender. For every position $j \notin J$, however, the honest receiver's internal iPAKE key K_j is independently uniform. This includes both positions at which an incorrect password test was made and positions at which no password-test query was issued. Moreover, these keys remain uniform even conditioned on the malicious sender's complete preceding view and its choice of the vector $(E_1, \ldots, E_n)$. Therefore, for every $x \in \mathcal{X}_{\mathsf{RRSS}}$, $\Pr[E_j \oplus K_j = x \,|\, E_1, \ldots, E_n,] = \frac{1}{|\mathcal{X}_{\mathsf{RRSS}}|}$ and these values are mutually independent across all $j \notin J$. To apply the malicious-dealer property formally, condition on the malicious sender's view and internal randomness immediately before the keys $(K_j)_{j \notin J}$ are sampled. Under this conditioning, the sender's vector $(E_1, \ldots, E_n)$ and the values at the positions in J are fixed arbitrarily, whereas every position outside J is independently replaced by a uniform element of $\mathcal{X}_{\mathsf{RRSS}}$. Since $|J| \leq t-1$, this is exactly the distribution considered in the $(t-1, \epsilon_{\mathsf{RRSS}}(\lambda))$ security against malicious dealers. Consequently, $\Pr[\mathsf{Identify}(\widetilde{s}_1, \ldots, \widetilde{s}_n) \neq \perp] \leq \epsilon_{\mathsf{RRSS}}(\lambda)$ By the definition of SecretRecover, except with this probability, the RRSS reconstruction algorithm returns $\perp$. In $\mathbf{H_7}$, when reconstruction returns $\perp$, the honest receiver outputs a fresh uniformly random session key. In $\mathbf{H_8}$, `Sim` instead marks the corresponding record as `interrupted` through the unsuccessful TestPwd query and submits $(\text{NewKey}, \mathsf{sid}, \mathsf{Party}_i, \perp)$ to $\mathcal{F}_{\mathsf{fPAKE}}$. Because the record is interrupted, the functionality ignores the supplied key and delivers a fresh independently uniform key to the honest receiver. Thus, the output distributions are identical unless the malicious-dealer security experiment fails. Therefore,

$$|\Pr[\mathbf{H_8}] - \Pr[\mathbf{H_7}]| \leq \epsilon_{\mathsf{RRSS}}(\lambda) \leq \mathsf{negl}(\lambda)$$

- **$\mathbf{H_9}$: Simulator does not give passwords to $\mathcal{F}_{\mathsf{iPAKE}}$ instances.** This hybrid also is the same as G_9 in [BFH+23]. In this hybrid, we modify the simulator as follows. Whenever $\mathtt{Sim}$ receives a message $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}_i, \mathsf{role})$ from $\mathcal{F}_{\mathsf{fPAKE}}$ for an honest party Party_i, it does not provide $\mathtt{pwd}_i$ as input to the internal $\mathcal{F}_{\mathsf{iPAKE}}$ instances. More precisely, instead of internally invoking $\mathcal{F}^j_{\mathsf{iPAKE}}$ with $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}_i[j], \mathsf{role})$ the simulator forwards $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathsf{role})$ to $\mathtt{Sim}^j_{\mathsf{iPAKE}}$, without providing the j-th character of $\mathtt{pwd}_i$. For each position $j \in [n]$, $\mathtt{Sim}$ checks whether this is the first $\textsc{NewSession}$ query for the corresponding internal iPAKE instance, or whether it is the second such query and a matching record for the other party already exists. In the latter case, $\mathtt{Sim}$ creates the corresponding internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ record for Party_i and marks it $\mathtt{fresh}$. Upon receiving a $\textsc{NewKey}$ query from an internal $\mathtt{Sim}^j_{\mathsf{iPAKE}}$, the simulator still emulates the behavior of $\mathcal{F}^j_{\mathsf{iPAKE}}$. In particular, $\mathtt{Sim}$ still uses the passwords received from $\mathcal{F}_{\mathsf{fPAKE}}$ to determine whether the two parties' j-th password characters match and, consequently, whether the corresponding internal instances must produce matching or independently uniform keys. The attacks on honest senders and honest receivers are handled as specified in $\mathbf{H_7}$ and $\mathbf{H_8}$, respectively. Thus, the only change is that the password character is no longer supplied as an explicit input to the internal $\mathcal{F}^j_{\mathsf{iPAKE}}$ instance. Since $\mathtt{Sim}$ itself produces exactly the output that $\mathcal{F}^j_{\mathsf{iPAKE}}$ would have produced for $\mathtt{Sim}^j_{\mathsf{iPAKE}}$, the view associated with every internal iPAKE instance remains unchanged and information theoretically the equivalent. Consequently:

  $$\Pr[\mathbf{H_9}] = \Pr[\mathbf{H_8}].$$

- **$\mathbf{H_{10}}$: Simulator makes honest and undisturbed parties output a key.** This hybrid is also the same as the $\mathbf{G_{10}}$ in [BFH+23]. In this hybrid, we modify both the dummy functionality and the simulator. First, we complete the transition from the dummy functionality to $\mathcal{F}_{\mathsf{fPAKE}}$ by modifying its response to $(\textsc{NewKey}, \mathsf{sid}, \mathsf{Party}_i, sk)$ queries from $\mathtt{Sim}$. For fresh records corresponding to two honest and undisturbed parties, the dummy functionality now behaves exactly as the $\textsc{NewKey}$ interface of $\mathcal{F}_{\mathsf{fPAKE}}$ defined in Figure 3. In particular, if $\mathtt{Ham}(\mathtt{pwd}_0, \mathtt{pwd}_1) \leq \ell$ the functionality samples a uniformly random session key and delivers the same key to both parties. Otherwise, it delivers independently uniform session keys. The remaining cases of the $\textsc{NewKey}$ interface, including records marked $\mathtt{compromised}$ or $\mathtt{interrupted}$, already behave as in Figure 3 by the changes introduced in $\mathbf{H_4}$.

  We next modify $\mathtt{Sim}$ so that it no longer uses the passwords of honest and undisturbed parties to determine their final session-key outputs. Upon receiving $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}_i, \mathsf{role})$ for such a party, $\mathtt{Sim}$ discards $\mathtt{pwd}_i$ and stores $\langle \mathsf{Party}_i, \mathsf{sid}, \mathsf{role}, \perp, \mathsf{MitM} = 0 \rangle$ instead of a record containing the password.

  The simulator does not change the answers delivered to the individual internal simulators $\mathtt{Sim}^j_{\mathsf{iPAKE}}$. Rather, it changes its reaction after receiving the $\textsc{NewKey}$ queries from all n internal simulators. These queries indicate that all per-character iPAKE executions have completed and that $\mathtt{Sim}$ can now generate the fPAKE-level message and trigger the party's final output. It proceeds according to the party's role as follows:

  - **Honest receiver.** If $\mathsf{role} = \mathsf{receiver}$, then $\mathtt{Sim}$ marks $\langle \mathsf{Party}_i, \mathsf{sid}, \mathsf{receiver}, \perp, \mathsf{MitM} = 0 \rangle$ as $\mathtt{waiting}$ and produces no output yet. Subsequently, upon receiving $(\textsc{Deliver}, \mathsf{sid}, \mathsf{Party}_{1-i}, \mathsf{Party}_i, \mathbf{E})$, $\mathtt{Sim}$ checks whether $\mathbf{E}$ is the message generated by the matching honest sender and whether it was delivered without modification. If so, $\mathtt{Sim}$ sends $(\textsc{NewKey}, \mathsf{sid}, \mathsf{Party}_i, \perp)$ to $\mathcal{F}_{\mathsf{fPAKE}}$. The receiver outputs the session key delivered by the functionality.
  - **Honest sender.** If $\mathsf{role} = \mathsf{sender}$, then $\mathtt{Sim}$ marks the corresponding record as $\mathtt{completed}$ and samples independent uniform per-character keys $K_1, \ldots, K_n$. It generates a valid RRSS sharing

$(s_1, \ldots, s_n)$ of a fresh uniformly random secret and encodes the shares using these keys exactly as an honest sender would; namely, it forms $\mathbf{E} = (E_1, \ldots, E_n)$, s.t. $E_j = s_j \oplus K_j$ for every $j \in [n]$. The simulator then sends $(\textsc{Send}, \mathsf{sid}, \mathsf{Party}_i, \mathsf{Party}_{1-i}, \mathbf{E})$ to $\mathcal{Z}$, formatted as a message from Party_i to Party_{1-i} through $\mathcal{F}_{\mathsf{SA}}$. Finally, $\mathtt{Sim}$ sends $(\textsc{NewKey}, \mathsf{sid}, \mathsf{Party}_i, \perp)$ to $\mathcal{F}_{\mathsf{fPAKE}}$, and the sender outputs the key delivered by the functionality.

These modifications affect only `fresh` records corresponding to two honest and undisturbed parties. Sessions involving a corrupted party or a man-in-the-middle attack are already handled by the changes in $\mathbf{H_6}$ and $\mathbf{H_8}$ through the `compromised` and `interrupted` branches of $\mathcal{F}_{\mathsf{fPAKE}}$.

We now compare $\mathbf{H_9}$ and $\mathbf{H_{10}}$. The distribution of the protocol message $\mathbf{E}$ is unchanged: in both hybrids, it is obtained by masking valid RRSS shares with independently uniform per-character keys. Thus, the change in this hybrid concerns only how the final outputs of honest and undisturbed parties are generated.

If $\mathtt{Ham}(\mathtt{pwd}_0, \mathtt{pwd}_1) \leq \ell$, the receiver obtains an RRSS sharing containing at most ℓ independently random corruptions. Therefore, by the correctness of Π_{RRSS} under random corruptions, the parties in $\mathbf{H_9}$ output the same session key except with probability $\epsilon_{\mathsf{RRSS}}(\lambda)$. This agrees with the matching keys delivered by $\mathcal{F}_{\mathsf{fPAKE}}$ in $\mathbf{H_{10}}$.

If $\mathtt{Ham}(\mathtt{pwd}_0, \mathtt{pwd}_1) > \ell$, then, for $t = n - \ell$, the receiver obtains at most $t - 1$ genuine RRSS shares, while all remaining recovered coordinates are independently uniform. By the $(t-1, \epsilon_{\mathsf{RRSS}}(\lambda))$ security of Π_{RRSS} against malicious dealers, $\mathsf{Identify}$ returns $\perp$ except with probability $\epsilon_{\mathsf{RRSS}}(\lambda)$. Hence, in $\mathbf{H_9}$, the receiver outputs a fresh key independent of the sender's key, exactly as prescribed by $\mathcal{F}_{\mathsf{fPAKE}}$ in $\mathbf{H_{10}}$, except when this RRSS security fails. Therefore, we have:

$$|\Pr[\mathbf{H_{10}}] - \Pr[\mathbf{H_9}]| \leq 2\ \epsilon_{\mathsf{RRSS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

- $\mathbf{H_{11}}$: **Simulator does not get inputs of the parties anymore** This hybrid is also the same as the $\mathbf{G_{11}}$ in [BFH+23]. In this hybrid, we modify the dummy ideal functionality so that it no longer reveals an honest party's password to $\mathtt{Sim}$. Specifically, instead of sending $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}_i, \mathsf{role})$ to $\mathtt{Sim}$, the functionality keeps $\mathtt{pwd}_i$ private and sends only $(\textsc{NewSession}, \mathsf{sid}, \mathsf{Party}_i, \mathsf{role})$. This change does not affect the execution because, following the modifications introduced in $\mathbf{H_{10}}$, $\mathtt{Sim}$ no longer uses $\mathtt{pwd}_i$ to simulate the n internal $\mathcal{F}_{\mathsf{iPAKE}}$ instances or determine the output of an honest and undisturbed party. In particular, the final session-key outputs of such parties are now determined by the $\textsc{NewKey}$ interface of $\mathcal{F}_{\mathsf{fPAKE}}$. Furthermore, as argued in $\mathbf{H_6}$ and $\mathbf{H_8}$, $\mathtt{Sim}$ does not use an honest party's password in sessions involving a corrupted party or in the presence of MitM attack. In these cases, $\mathtt{Sim}$ either uses independently uniform keys $K_1, \ldots, K_n$ or learns, through $\textsc{TestPwd}$ queries to $\mathcal{F}_{\mathsf{fPAKE}}$, which password-character guesses are correct and, consequently, which internal iPAKE sessions should produce matching keys. Thus, the simulator requires only the permitted leakage from $\mathcal{F}_{\mathsf{fPAKE}}$ and never the password itself. Therefore, the password removed from the $\textsc{NewSession}$ notification is not used anywhere in the simulation. The views of $\mathcal{Z}$ in $\mathbf{H_{10}}$ and $\mathbf{H_{11}}$ are consequently information theoretically equivalent and distributed identically, and we have:

$$\Pr[\mathbf{H_{11}}] = \Pr[\mathbf{H_{10}}].$$

We note that the ideal functionality in hybrid $\mathbf{H_{11}}$ is exactly the same as $\mathcal{F}_{\mathsf{fPAKE}}$ in Figure 3 with the simulators described in Figure 4 and Figure 5. This in fact concludes our security proof.

Putting indistinguishability of all these hybrids and applying union bound we have

$$\begin{aligned}|\Pr[H_{11}] - \Pr[H_0]| &\leq \sum_{i=1}^{11} |\Pr[H_i] - \Pr[H_{i-1}]| \\ &\leq 2n\epsilon_{\mathsf{iPAKE}} + 2\epsilon_{\mathsf{RRSS}}(\lambda) + 3\mathsf{negl}(\lambda) \leq \mathsf{negl}(\lambda)\end{aligned}$$

Simulator Sim for $\mathcal{F}_{\mathsf{fPAKE}}$ (Part I) **On** (NewSession, sid, P_i, role) from $\mathcal{F}_{\mathsf{fPAKE}}$ (or (Init, $\langle P_0, P_1\rangle$, sid) from a corrupted P_i to $\mathcal{F}_{\mathsf{SA}}$):

- Create record $(P_i, \mathsf{sid}, \mathsf{role}, \perp)$ and mark it fresh.
- Send (Init, $\langle P_0, P_1\rangle$, sid) to $\mathcal{Z}$ in the name of $\mathcal{F}_{\mathsf{SA}}$.

On (Init, sid, P_i, H, $\mathsf{sid}_{\mathbb{H}}$) from $\mathcal{Z}$ in the name of $\mathcal{A}$ to $\mathcal{F}_{\mathsf{SA}}$:

- Perform the same checks as $\mathcal{F}_{\mathsf{SA}}$ on sid_H and H and ignore the message if one fails.
- If P_i is corrupted then provide output (Init, sid, sid_H) to P_i.
- If $\mathbb{H} = \{P_0, P_1\}$ then set MitM := 0, else MitM := 1.
- Retrieve $(P_i, \mathsf{sid}, \mathsf{role}, \perp)$ and replace $\perp$ by MitM.
- If there is no instance of $\mathcal{F}_{\mathsf{iPAKE}}$ for session id $(\mathsf{sid}, \mathsf{sid}_H, j)$ for $j \in [n]$, create one.
- For all $j \in [n]$ send (NewSession, $(\mathsf{sid}, \mathsf{sid}_H, j)$, P_i, role) to $\mathcal{F}_{\mathsf{iPAKE}}$.

On receiving output Z_j from all n instances of $\mathcal{F}_{\mathsf{iPAKE}}$:

- Store $(\mathsf{sid}, P_i, P_{1-i}, \mathrm{MSG} := (Z_1, \ldots, Z_n)$.

On instruction from $\mathcal{Z}$ to send (Deliver, sid, P_i, P_{1-i}, msg) to $\mathcal{F}_{\mathsf{SA}}$:

- If MitM = 0:
  - If msg is the one sent by Sim, instruct $\mathcal{F}_{\mathsf{iPAKE}}$ to send it.
  - Remove one occurrence of the stored record.
- Else (MitM = 1): give the message to $\mathcal{Z}$.

On (TestPwd, $(\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}, j)$, P_i, $\mathtt{pwd}_j$) from $\mathcal{F}_{\mathsf{iPAKE}}$:

- Store (TestPwd, sid, P_i, j, $\mathtt{pwd}_j$).

On instruction from $\mathcal{Z}$ to send (Deliver, sid, P_{1-i}, P_i, E') to $\mathcal{F}_{\mathsf{SA}}$:

- Retrieve record $(P_i, \mathsf{sid}, \mathsf{role}, \mathrm{MitM}, K)$ marked waiting.
- If MitM = 0 and both parties are honest:
  - Remove the record.
  - If role = receiver then send (NewKey, sid, P_i, $\perp$) to $\mathcal{F}_{\mathsf{fPAKE}}$.
- Else if P_i is honest and role = receiver:
  - Compute $s' := \Pi_{\mathsf{RRSS}}.\mathsf{SecretRecover}(\vec{E'} \oplus \vec{K})$.
  - If $s' \neq \perp$, then output key sk accordingly.
  - Else output $\perp$.
- Else if P_i is corrupt, output (Received, sid, P_{1-i}, P_i, E') to P_i.

Random oracle queries

- On query $(\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}, s)$ to $\mathcal{H}$: return stored value or sample $k \leftarrow \{0,1\}^{\lambda}$.

Figure 4: The first part of the simulator Sim for $\mathcal{F}_{\mathsf{fPAKE}}$.

and our fPAKE protocol described in Figure 1 UC-realizes the ideal functionality $\mathcal{F}_{\mathsf{fPAKE}}$ described in Figure 3.

□

G Review of Cryptographic Tools and Building Blocks

G.1 Shamir Secret Sharing

Let $\mathbb{F}$ be a field of size $|\mathbb{F}| \geq n$. We define interpolation algorithm InterPol over t-degree polynomial $f : \mathbb{F} \to \mathbb{F}$ which takes as input t tuples $(x_i = i, y_i = f(i))$ for $1 \leq i \leq t$, and outputs the main secret $f(0)$. In particular, we have $f(x) = \sum_{i \in S} y_i L_{i,S}(x)$ where $L_{i,S}(x) = \prod_{j \in S \setminus \{i\}} \left(\frac{x - x_i}{x_j - x_i}\right)$ denotes the Lagrange interpolating polynomial which can be computed. Finally, the original secret can be obtained by computing $f(0)$. For simplicity, we use $\mathsf{InterPol}((x_1, f(x_1)), \ldots, (x_t, f(x_t)))$ to denote the Lagrage interpolation. The (n, t)- secret sharing scheme for the secret message $s \in \mathbb{F}$ is defined based on two algorithms ShareGen, SecretRecover which are described in what follows.

- $([\![s]\!]_1, \ldots, [\![s]\!]_n) \leftarrow \mathsf{ShareGen}(n, t, s)$. This algorithm takes as input the secret $s \in \mathbb{F}$, the threshold value t and the number of shares n. Then, it randomly samples the $(t-1)$-degree polynomial $\phi : \mathbb{F} \to \mathbb{F}$ such that $s = \phi(0)$ and sets the shares as $[\![s]\!]_i = \phi(i)$, for all $1 \leq i \leq n$. To randomly sample the $t-1$-degree polynomial $\phi(x) = s + a_1 x + a_2 x^2 + \ldots + a_{t-1} x^{t-1}$, we simply select $\{a_i \in_R \mathbb{F}\}_{\forall i \in [t]}$ uniformly at random and form $\phi(x) = s + \sum_{i=1}^{t-1} a_i x^i$.
- $s \leftarrow \mathsf{SecretRecover}(i_1, \ldots, i_t, [\![s]\!]_{i_1}, \ldots, [\![s]\!]_{i_t})$. This algorithm takes as input a list $1 \leq i_1 < \ldots < i_t \leq n$ of t distinct indices along with their corresponding shares $[\![s]\!]_{i_1}, \ldots, [\![s]\!]_{i_t}$. To recover the secret s the algorithm simply runs the interpolation algorithm $s = \mathsf{InterPol}((i_1, [\![s]\!]_{i_1}), \ldots, (i_t, [\![s]\!]_{i_t}))$.

Simulator $\mathtt{Sim}$ for $\mathcal{F}_{\mathsf{fPAKE}}$ (Part II)

On $(\textsc{NewKey}, \mathsf{sid}, P_i, sk_j)$ from $\mathtt{Sim}^{j}_{\mathsf{fPAKE}}$: record $\langle \textsc{NewKey}, \mathsf{sid}, P_i, sk_j \rangle$. If this is the n-th $\textsc{NewKey}$ query for P_i, do:

- Retrieve record $\langle P_i, \mathsf{sid}, \mathsf{role}, \textsc{MitM} \rangle$ marked $\mathtt{fresh}$.
- If $\textsc{MitM} = 0$ and both parties of session sid are honest: choose $K \leftarrow \mathbb{F}^n_{2^\lambda}$.
- Else // $\textsc{MitM} = 1$ or one party is corrupted
  - Retrieve records $\langle \textsc{TestPwd}, \mathsf{sid}, P_i, j, \mathtt{pwd}'_j \rangle$ for any $j \in [n]$ where such a record exists. Let J be the set of j for which a record exists.
  - If $|J| \geq n - \ell$ such records, choose $\mathtt{pwd}'_t \leftarrow \Sigma$ for $t \in [n] \setminus J$ and send $(\textsc{TestPwd}, \mathsf{sid}, P_i, \mathtt{pwd}'_1 \| \cdots \| \mathtt{pwd}'_n)$ to $\mathcal{F}_{\mathsf{fPAKE}}$.
  - Else // $|J| < n - \ell$ choose $K \leftarrow \mathbb{F}^n_{2^\lambda}$ and send $(\textsc{TestPwd}, \mathsf{sid}, P_i, \perp)$ to $\mathcal{F}_{\mathsf{fPAKE}}$. // interrupt the record
  - Regardless, wait for $\mathcal{F}_{\mathsf{fPAKE}}$'s response:
    - ♠ If $\mathcal{F}_{\mathsf{fPAKE}}$ responds with $(T, \text{“correct guess”})$, check if $|T \cap J| < n - \ell$. If so, mark $\langle P_i, \mathsf{sid}, \mathsf{role}, \textsc{MitM} \rangle$ as $\textsc{Accident}$. Else retrieve records $\langle \textsc{NewKey}, \mathsf{sid}, P_i, sk_t \rangle$ and set $K_t := sk_t$ for all $t \in T \cap J$ and choose $K_t \leftarrow \mathbb{F}_{2^\lambda}$ for all $t \in [n] \setminus (T \cap J)$. Mark the record $\langle P_i, \mathsf{sid}, \mathsf{role}, \textsc{MitM} \rangle$ as $\mathtt{compromised}$.
    - ♠ Else // $\mathcal{F}_{\mathsf{fPAKE}}$ responds with “wrong guess” choose $\vec{K} \leftarrow \mathbb{F}^n_{2^\lambda}$ and mark the record $\langle P_i, \mathsf{sid}, \mathsf{role}, \textsc{MitM} \rangle$ as $\mathtt{interrupted}$.
- Regardless:
  - If $\mathsf{role} = \textsc{Receiver}$ then append $\vec{K}$ to the record $\langle P_i, \mathsf{sid}, \mathsf{role}, \textsc{MitM} \rangle$ and mark it $\mathtt{waiting}$.
  - Else // $\mathsf{role} = \textsc{Sender}$
    - ♠ Choose $s \leftarrow \{0,1\}^\lambda$ and $\mathsf{pp} \leftarrow \Pi_{\mathsf{RRSS}}.\mathsf{Setup}(1^\lambda)$, and set $C := \Pi_{\mathsf{RRSS}}.\mathsf{ShareGen}(s, \mathsf{pp})$, and $\vec{E} := \vec{C} \oplus \vec{K}$.
    - ♠ If the record $\langle P_i, \mathsf{sid}, \mathsf{role}, \textsc{MitM} \rangle$ is marked $\textsc{Accident}$, choose $\widehat{sk} \leftarrow \{0,1\}^\lambda$.
    - ♠ Else if it is marked $\mathtt{compromised}$, set $\widehat{sk} := \mathcal{H}((\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}), s)$.
    - ♠ Else // $\mathtt{interrupted}$ or $\mathtt{fresh}$ set $\widehat{sk} := \perp$.
    - ♠ Regardless, send $(\textsc{NewKey}, \mathsf{sid}, P_i, \widehat{sk})$ to $\mathcal{F}_{\mathsf{fPAKE}}$. Store $\langle \mathsf{sid}, P_i, P_{1-i}, \vec{E} \rangle$ and send $(P_i, P_{1-i}, \vec{E})$ to $\mathcal{Z}$ as a message from $\mathcal{F}_{\mathsf{SA}}$. Mark $\langle P_i, \mathsf{sid}, \mathsf{role}, \textsc{MitM} \rangle$ as $\mathsf{completed}$.

Figure 5: The second part of the simulator $\mathtt{Sim}$ for $\mathcal{F}_{\mathsf{fPAKE}}$ and our Fuzzy PAKE with Random Robust Secret Sharing. $\mathtt{Sim}$ simulates $\mathcal{F}_{\mathsf{SA}}$ and the random oracle $\mathcal{H}$. If we write “retrieve record” and the record does not exist, $\mathtt{Sim}$ ignores the message.

It is easy to verify that the (t, n)-secret sharing scheme $\Pi = (\mathsf{ShareGen}, \mathsf{SecretRecover})$ described above is perfectly correct and provides $t-1$-perfect privacy. In Lemma 2 we formally prove that Shamir Secret Sharing provides $t-1$-perfect privacy. We stress that we are not claiming that this result is new or novel. However, for completeness, we provide an independent proof here in the appendix[7].

Lemma 2 ($t-1$-Perfect privacy for Shamir Secret Sharing) *(n, t)-Shamir secret sharing scheme provides $(t-1)$-perfect privacy.*

Proof of Lemma 2: Let $\mathbb{F}$ be a finite field, let the secret be $s \in \mathbb{F}$, and let$\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}$ be distinct nonzero public evaluation points. In a (t, n)-Shamir secret sharing scheme, the dealer samples $(a_1, a_2, \ldots, a_{t-1}) \xleftarrow{\$} \mathbb{F}^{t-1}$ independently and uniformly at random, and defines the polynomial $f(z) = s + a_1 z + a_2 z^2 + \cdots + a_{t-1} z^{t-1}$. The i-th share is then $y_i = f(\alpha_i)$. We want to prove that the joint distribution of any subset of at most $t-1$ shares is uniform over the appropriate vector space, independently of the secret $s \in \mathbb{F}$. More precisely, for every $k \leq t-1$, for every subset of indices $\{i_1, \ldots, i_k\} \subseteq [n]$ and for every vector $(x_1, \ldots, x_k) \in \mathbb{F}^k$ we will prove that for any fixed s

$$\Pr\left[(y_{i_1}, \ldots, y_{i_k}) = (x_1, \ldots, x_k)\right] = \frac{1}{|\mathbb{F}|^k}$$

where $y_i = f(\alpha_i)$ for each i and the randomness is taken over the selection of the coefficients $\alpha_1, \ldots, \alpha_t$.

The proof of this lemma has two steps. In the first step we first translate our target event to its equivalent linear system of equations. Then, we compute the rand of the resulting matrix. In the third step of the proof, we count the number of all possible solutions to our system of linear equations. Finally, we conclude the proof by computing the probability of the event we plan to compute.

Step 1: Translate the event into a linear system. Fix a secret $s \in \mathbb{F}$, fix distinct indices $i_1, \ldots, i_k$, and fix target values $x_1, \ldots, x_k \in \mathbb{F}$. The event $(y_{i_1}, \ldots, y_{i_k}) = (x_1, \ldots, x_k)$ means exactly that $f(\alpha_{i_r}) =$

[7]This proof was developed with the assistance of ChatGPT for drafting and exposition. The authors have verified all arguments and take full responsibility for the correctness and content of the result.

x_r for each $r = 1, \ldots, k$. Substituting the definition of f, this becomes $s + a_1\alpha_{i_r} + a_2\alpha_{i_r}^2 + \cdots + a_{t-1}\alpha_{i_r}^{t-1} = x_r$ for $r = 1, \ldots, k$ Equivalently, $a_1\alpha_{i_r} + a_2\alpha_{i_r}^2 + \cdots + a_{t-1}\alpha_{i_r}^{t-1} = x_r - s$ for $r = 1, \ldots, k$.

Thus, the event that the selected shares equal $(x_1, \ldots, x_k)$ is equivalent to saying that the random coefficient vector $A = (a_1, \ldots, a_{t-1})^\top \in \mathbb{F}^{t-1}$ satisfies a linear system $MA = b$ where $M = \begin{pmatrix} \alpha_{i_1} & \alpha_{i_1}^2 & \cdots & \alpha_{i_1}^{t-1} \\ \alpha_{i_2} & \alpha_{i_2}^2 & \cdots & \alpha_{i_2}^{t-1} \\ \vdots & \vdots & & \vdots \\ \alpha_{i_k} & \alpha_{i_k}^2 & \cdots & \alpha_{i_k}^{t-1} \end{pmatrix} \in \mathbb{F}^{k\times(t-1)}$ and $b = \begin{pmatrix} x_1 - s \\ x_2 - s \\ \vdots \\ x_k - s \end{pmatrix} \in \mathbb{F}^k$.

Step 2: Show that the matrix M has rank k. The matrix M is a rectangular Vandermonde-type matrix. Since the evaluation points $\alpha_{i_1}, \ldots, \alpha_{i_k}$ are distinct and nonzero, its rows are linearly independent.

A convenient way to see this is to factor one α_{i_r} from each row:$(\alpha_{i_r}, \alpha_{i_r}^2, \ldots, \alpha_{i_r}^{t-1}) = \alpha_{i_r}(1, \alpha_{i_r}, \ldots, \alpha_{i_r}^{t-2})$. Because each $\alpha_{i_r} \neq 0$, multiplying a row by α_{i_r} does not affect linear independence. Therefore the row rank of M is the same as the row rank of $M' = \begin{pmatrix} 1 & \alpha_{i_1} & \cdots & \alpha_{i_1}^{t-2} \\ 1 & \alpha_{i_2} & \cdots & \alpha_{i_2}^{t-2} \\ \vdots & \vdots & & \vdots \\ 1 & \alpha_{i_k} & \cdots & \alpha_{i_k}^{t-2} \end{pmatrix}$. Now look at the first k columns of M'. They form the $k \times k$ Vandermonde matrix $V = \begin{pmatrix} 1 & \alpha_{i_1} & \alpha_{i_1}^2 & \cdots & \alpha_{i_1}^{k-1} \\ 1 & \alpha_{i_2} & \alpha_{i_2}^2 & \cdots & \alpha_{i_2}^{k-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & \alpha_{i_k} & \alpha_{i_k}^2 & \cdots & \alpha_{i_k}^{k-1} \end{pmatrix}$. Its determinant is $\det(V) = \prod_{1\le r<\ell\le k}(\alpha_{i_\ell} - \alpha_{i_r})$, which is nonzero because the α_{i_r} are distinct. Hence V is invertible, so its rows are linearly independent. Therefore the rows of M' are linearly independent, and hence the rows of M are linearly independent. We conclude that $\mathsf{rank}(M) = k$.

Step 3: Count the number of solutions to $MA = b$. The unknown vector A has $t - 1$ coordinates, so the ambient space is $\mathbb{F}^{t-1}$. Since M has rank k, then we can solve the system of linear equations for k variables and the remaining $t - 1 - k$ variables are free. Each free variable has $|\mathbb{F}|$ possible cases. So, all possible solutions is $|\mathbb{F}|^{t-1-k}$.

Finally the probability $\Pr[y_1, \ldots, y_k = x_1, \ldots, x_k|s]$ is equivalent to the ratio of all valid solutions for our system of linear equation to the all possible values for the vector $(a_1, \ldots, a_{t-1})$ and we have:

$$\Pr[y_1, \ldots, y_k = x_1, \ldots, x_k] = \frac{|\mathbb{F}|^{t-1-k}}{|\mathbb{F}^{t-1}|} = \frac{1}{|\mathbb{F}^k|}$$

Since the above equations is correct for all $k \le t - 1$, so we have:

$$\Pr[y_1, \ldots, y_k = x_1, \ldots, x_k] = \frac{1}{|\mathbb{F}^{t-1}|}$$

and the joint distribution of any subset of $t - 1$ shares is uniform.

Conclusion. We have shown that for every fixed secret $s \in \mathbb{F}$, every subset of indices $\{i_1, \ldots, i_k\}$ with $k \leq t-1$, and every target vector $(x_1, \ldots, x_k) \in \mathbb{F}^k$,

$$\Pr[(y_{i_1}, \ldots, y_{i_k}) = (x_1, \ldots, x_k)] = \frac{1}{|\mathbb{F}|^k} .$$

□

G.2 Details on Paillier Cryptosystem

In this part we will provided a formal definition of Paillier cryptosystem with more details. We should highlight that the Paillier cryptosystem [Pai99] consists of three main algorithms $\Pi_{\mathtt{P}} = (\mathtt{P.KeyGen}, \mathtt{P.Enc}, \mathtt{P.Dec})$, and two main operations $\mathtt{P.Add}, \mathtt{P.PlainToCtxMul}$ which are described as follows.

- $(pk, sk) \leftarrow \mathtt{P.KeyGen}\left(1^\lambda; (p, q)\right)$: This algorithm takes as input the security parameter λ and two uniformly at random sampled large $\mathsf{poly}(\lambda)$ bit prime numbers $p, q \leq 2^{\mathsf{poly}(\lambda)}$. We should highlight that based on the security parameter we desire, there exists a polynomial function like poly which determined the bit length of the prime numbers. If the values of p and q are not large enough to provide λ-bit security level, we resample the values of prime numbers p and q. Then, the algorithm generates the secret-public key pair (pk, sk) as follows. It sets $N = pq$ and computes $\beta = \mathsf{lcm}(p-1, q-1)$. Let we define the function $L(u) = \frac{u-1}{N}$ in which $u \in [N^2]$ is a variable. So using the defined function to compute $\mu = (L(g^\beta \mod N^2))^{-1} \mod N$ in which $g \in \mathbb{Z}^*_{N^2}$ is sampled uniformly at random. Finally, the public key and its corresponding secret key is set as follows: $pk = (N, g)$ and $sk = (\beta, \mu)$. In this paper we considered $g = N + 1$. In this case, $\beta = \mathsf{lcm}(p-1, q-1)$ and $\mu = \Phi(N)^{-1} \mod N$.
- $c \leftarrow \mathtt{P.Enc}_{pk}(m; r)$: The encryption algorithm takes as input the message $0 \leq m < N$ and random coin $r \in_R \mathbb{Z}^*_N$, and computes the ciphertext as follows: $c := (1+N)^m r^N \mod (N^2)$.
- $m := \mathtt{P.Dec}_{sk}(c)$: The decryption algorithm takes as input the ciphertext $c \in \mathbb{Z}^*_{N^2}$ and decrypts it as follows: $m := L(c^\beta \mod N^2).\mu \mod N$.
- $c = \mathtt{P.Add}(c_1, c_2)$: Given $c_1 = \mathtt{P.Enc}_{pk}(m_1)$ and $c_2 = \mathtt{P.Enc}_{pk}(m_2)$ for two messages m_1, m_2, this algorithm computes a ciphertext of $m_1 + m_2 \mod N$ under the same public keys as follows. $c = c_1.c_2 \mod N^2$. Intuitively we have:

$$\begin{aligned} c &= (g^{m_1} r_1^N).(g^{m_2} r_2^N) \mod N^2 \\ &= g^{m_1+m_2}(r_1 r_2)^N \mod N^2 \\ &= g^m r^N \mod N^2 \end{aligned} \tag{1}$$

  in which $m = m_1 + m_2$ and the resulting randomness is $r = r_1 r_2$.

  For sake of simplicity, we use symbol $\boxplus$ for this algorithm and we have $c_1 \boxplus c_2 = \mathtt{P.Add}(c_1, c_2)$. Similarly, for the subtraction, we can define $\boxminus$ and we have $c_1 \boxminus c_2 = \mathtt{P.Add}(c_1, c_1^{-1})$. We can also define $\uplus$ which add k ciphertexts $c_1, \ldots, c_k$ and we have $\uplus_{i=1}^k c_i = c_1 \boxplus \ldots \boxplus c_k$.
- $c = \mathtt{P.PlainToCtxMul}(m_1, c_2)$: This algorithm takes as input the plaintext message m_1 and $c_2 = \mathtt{P.Enc}_{pk}(m_2)$, and computes ciphertext of $m_1.m_2$ as follows:

$$\begin{aligned} c &= (c_2)^{m_1} \mod N^2 = (g^{m_2} r_2^N)^{m_1} = g^{m_1 m_2}(r_2^{m_1})^N \mod N^2 \\ &= g^m r^N \mod N^2 \end{aligned} \tag{2}$$

  in which $m = m_1 m_2$ and the resulting randomness is $r = r_2^{m_1} \mod N^2$.

  For sake of simplicity, we use symbol $\boxtimes$ for this algorithm and we have $m_1 \boxtimes c_2 = \mathtt{P.PlainToCtxMul}(m_1, c_2)$.

G.3 Implicit-only PAKE (iPAKE)

Both [BFH+23] and [DHP+18] constructions used iPAKE as the main building block of their construction. An iPAKE is a specific type of PAKE which only achieves implicit authentication with respect to the honest parties and the adversary. In other words, the implicit terminology implies that in the end of the protocol, the two involved parties may not know if they extracted the same key or not. In addition, an active adversary who executes the attack on the protocol by guessing the password does not get any feedback on the correctness of their guess. We follow the ideal functionality $\mathcal{F}_{\mathsf{iPAKE}}$ defined by Dupont et al. [DHP+18] which is described in Figure 6. We should point out that Roy and Xu [RX23] showed that the ideal UC-functionality for PAKE like [CHK+05] do not offer correctness. Intuitively, the two parties can securely realize the PAKE functionality when the parties always output random keys. They showed that it is impossible to incorporate the correctness into ideal functionality, without considering the Man-In-the-Middle Attack. Additionally, they proved that this formulation is equivalent to demanding correctness from the PAKE protocol as a separate property. Similar to [BFH+23] we also require the correctness for iPAKE as well and is defined as follows.

The ideal functionality $\mathcal{F}_{\mathsf{iPAKE}}$ where λ is the security parameter. It interacts with two involved parties $\mathsf{Party}_0, \mathsf{Party}_1$ and the adversary $\mathcal{A}$ via the following queries:

- **Upon receiving a query** $(\textbf{NewSession}, \mathsf{sid}, \mathtt{pwd}_i, \mathsf{role})$ **from** Party_i**:**
  - Send $(\text{NewSession}, \mathsf{sid}, \mathsf{Party}_1, \mathsf{role})$ to $\mathcal{A}$.
  - If one of the following is true, record $(\mathsf{Party}_i, \mathtt{pwd}_i)$ and mark this record as `fresh`:
    - This is the first NewSession query.
    - This is the second NewSession query and there is a record $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$.
- **Upon receiving a query** $(\textbf{TestPwd}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}^*)$ **from** $\mathcal{A}$**:**

  If there is a `fresh` record $(\mathsf{Party}_i, \mathtt{pwd}_i)$, then:
  - If $\mathtt{pwd}_i = \mathtt{pwd}^*$, mark the record `compromised`;
  - If $\mathtt{pwd}_i \neq \mathtt{pwd}^*$, mark the record as `interrupted`;
- **Upon receiving query** $(\textbf{NewKey}, \mathsf{sid}, \mathsf{Party}_i, K)$ **from** $\mathcal{A}$**, where** $|K| = \lambda$**:**

  If there is no record of the form $(\mathsf{Party}_i, \mathtt{pwd}_i)$, or if this is not the first NewKey query for Party_i, then ignore this query; otherwise:
  - If at least one of the following is true, then output (sid, K) to player Party_i:
    - The record is `compromised`.
    - Party_i is corrupted.
    - The record is `fresh`, Party_{1-i} is corrupted, and there is a record $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$ with $\mathtt{pwd}_i = \mathtt{pwd}_{i-1}$.
  - If this record is `fresh`, both parties are honest, there is a record $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$ with $\mathtt{pwd}_i = \mathtt{pwd}_{1-i}$, a key K' was sent to Party_{1-i}, and $(\mathsf{Party}_{1-i}, \mathtt{pwd}_{1-i})$ was `fresh` at the time, then output (sid, K') to Party_i;
  - In any other case, pick a new random key K' of length λ and send (sid, K') to Party_i.
  - Mark the record $(\mathsf{Party}_i, \mathtt{pwd}_i)$ as `completed`.

Figure 6: Functionality $\mathcal{F}_{\mathsf{iPAKE}}$ [DHP+18]

Definition 5 (iPAKE Correctness [BFH+23]) *We say an iPAKE protocol π is ϵ-correct if for all $\mathtt{pwd} \in \mathcal{PWD}$, for some finite dictionary $\mathcal{PWD}$, and for all passive (i.e., honest-but-curious) adversaries $\mathcal{A}$, the probability that in an execution of π with honest parties Party_0, Party_1 the first party Party_0 on input $\mathtt{pwd}$ outputs a different key than the second party Party_1 on input $\mathtt{pwd}$ in the presence of $\mathcal{A}$ is at most $\epsilon(\lambda)$ and we have*

$$\Pr[K \neq K' | (K, K') \leftarrow out_{\mathcal{A},\pi}\langle \mathsf{Party}_0(\mathtt{pwd}), \mathsf{Party}_1(\mathtt{pwd})\rangle] \leq \epsilon(\lambda)$$

where the probability is taken over the random coin of $\mathsf{Party}_0, \mathsf{Party}_1$ *and* $\mathcal{A}$ *and the protocol* π*. The notation* $(K, K') \leftarrow out_{\mathcal{A},\pi}\langle \mathsf{Party}_0(\mathtt{pwd}), \mathsf{Party}_1(\mathtt{pwd})$ *implies the output of the protocol* π *in presence of adversary* $\mathcal{A}$ *where* K *and* K *are given to* Party_0 *and* Party_1*, respectively.*

G.4 Split Authentication

Another building used in the fPAKE construction in [BFH+23] is *Split Authentication*, introduced by Barak et al. [BCL+05]. Essentially, a split authentication (SA) is a protocol which implements something close to an authenticated channel in an unauthenticated setting. In this setting, the adversary has an additional ability to run two separate executions of an authentication scheme without (one with each party) without the two parties realizing it. In the case that the parties are both honest, the adversary in the beginning of the protocol has the option of deciding to launch a man-in-the-middle attack and execute separate protocols with each party or to just observe the exchanged messages or add adversarial delay to the message delivery. The ideal functionality $\mathcal{F}_{\mathsf{SA}}$ for split authentication is formalized by Barak et al. [BCL+05] and we recall it in Figure 7.

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

Initialization:

- **Activation**: Upon activation with input $(\textsc{Init}, \mathsf{sid})$, where $\mathsf{sid} = (\mathbb{P}, \mathsf{sid}')$ and $\mathbb{P}$ is the set of PIDs: record $\mathbb{P}$ and forward it to the adversary.
- **Registration**: On $(\textsc{Init}, \mathsf{sid}, \mathsf{Party}, \mathbb{H}, \mathsf{sid}_{\mathbb{H}})$ from the adversary, verify that:
  - $\mathsf{Party} \in \mathbb{P}$ and $\mathsf{Party} \in \mathbb{H}$.
  - for all recorded sets $\mathbb{H}'$, either
    - $\mathbb{H} \cap \mathbb{H}'$ contains only corrupt parties and $\mathsf{sid}_{\mathbb{H}} \neq \mathsf{sid}_{\mathbb{H}'}$ or
    - $\mathbb{H}' = \mathbb{H}$ and $\mathsf{sid}_{\mathbb{H}} = \mathsf{sid}_{\mathbb{H}'}$.

  If so, then output $(\textsc{Init}, \mathsf{sid}, \mathsf{sid}_{\mathbb{H}})$ to Party and record $(\mathbb{H}, \mathsf{sid}_{\mathbb{H}})$ if not yet recorded. Else, do nothing.

Message Authentication:

- **Send**: On input $(\textsc{Send}, \mathsf{sid}, \mathsf{Party}, \mathsf{Party}', msg)$ from Party, where $\mathsf{Party}' \in \mathbb{P}$, send $(\mathsf{Party}, \mathsf{Party}', msg)$ to $\mathcal{A}$ and add $(\mathsf{Party}, \mathsf{Party}', msg)$ to an (initially empty) list $\mathbb{W}$ of waiting messages. Note that the same triple can be appear multiple times in the list. Else, do nothing.
- **Deliver**: On $(\textsc{Deliver}, \mathsf{sid}, \mathsf{Party}, \mathsf{Party}', msg)$ from $\mathcal{A}$, proceed as follows:
  - If Party' did not previously receive an $(\textsc{Init}, \mathsf{sid}, \cdot)$, then do nothing.
  - Else, if Party is in the authentication set $\mathbb{H}$ of Party', and Party is honest, then do:
    - ♠ If there is a triple $(\mathsf{Party}, \mathsf{Party}', msg) \in \mathbb{W}$, then remove one appearance of the triple from $\mathbb{W}$ and output $(\textsc{Received}, \mathsf{sid}, \mathsf{Party}, \mathsf{Party}', msg)$ to Party'.
    - ♠ Otherwise, do nothing.
  - Else (i.e., Party' received $(\textsc{Init}, \mathsf{sid}, \cdot)$, and either Party is corrupted or $\mathsf{Party} \notin \mathbb{H}$), output $(\textsc{Received}, \mathsf{sid}, \mathsf{Party}, \mathsf{Party}', msg)$ to Party', regardless of $\mathbb{W}$.

Figure 7: Ideal functionality $\mathcal{F}_{\mathsf{SA}}$ for split authentication [BCL+05].

G.5 Global Random Oracle Model

Here we described the ideal functionality of Global random oracle model [CJS14] $\mathcal{F}_{GRO}$ which is UC-realized using random oracle model. Basically, we can model the random oracle as a global setup functionality $\mathcal{F}_{\mathsf{GRO}}$ that is accessible to all parties and sessions. The functionality maintains a list of past queries and their responses, ensuring consistency across invocations. Upon receiving a query x, it returns a uniformly random string of length $\ell(n)$ if x has not been queried before, and otherwise returns the previously assigned value.

Functionality $\mathcal{F}_{\text{GRO}}$

Parameters: Output length $\ell(n)$ and a list $\bar{\mathcal{F}}$ of ideal functionality programs.

State: An initially empty list Q of query–response pairs and, for each session identifier sid, a list Q_{sid} of illegitimate queries.

(1) Upon receiving a query x from some party $P = (\mathsf{pid}, \mathsf{sid})$ or from the adversary S:
- If $(x, v) \in Q$ for some $v \in \{0,1\}^{\ell(n)}$, return v to P.
- Otherwise, sample $v \leftarrow \{0,1\}^{\ell(n)}$ uniformly at random, store (x, v) in Q, and return v to P.

Parse x as (sid, x'). If $\mathsf{sid} \neq \perp$, add (sid, x', v) to the list Q_{sid} of illegitimate queries.

(2) Upon receiving a request from an instance of an ideal functionality in $\bar{\mathcal{F}}$ with session identifier sid, return the list Q_{sid}.

Figure 8: Global Random Oracle functionality $\mathcal{F}_{\mathsf{GRO}}$. [CJS14]

Queries made by the adversary are additionally recorded as illegitimate queries. This global functionality enables all protocol instances to share the same oracle while preserving the unpredictability and consistency properties of a random oracle in the UC framework.

H Conditional Encryption

In this section, we review formal definition of a semi-honest *conditional encryption* scheme as introduced in [AB24]. In comparison to the traditional public key encryption schemes, a conditional encryption scheme contains similar algorithms with the addition of a special algorithm CEnc_{pk} called conditional encryption. This algorithm takes as inputs: a ciphertext $c = \mathsf{Enc}_{pk}(m_1; r) \in \mathcal{C}_{\mathsf{Enc}}$ (Where $\mathcal{C}_{\mathsf{Enc}}$ is the ciphertext space of traditional encryption scheme) encrypting some unknown message $m_1 \in \mathcal{M}$ in our message space using random coin $r \in_R \{0,1\}^\lambda$, a control message m_2 and a payload message m_3. A conditional encryption scheme is defined with respect to a binary predicate $P : \mathcal{M} \times \mathcal{M} \to \{0,1\}$. Intuitively, if $P(m_1, m_2) = 1$, then $\mathsf{CEnc}_{pk}(c, m_2, m_3) \in \mathcal{C}_{\mathsf{CEnc}}$ (Similarly, we define $\mathcal{C}_{\mathsf{CEnc}}$ as the ciphertext space for a conditional encryption scheme) should produce valid encryption of our payload message m_3; otherwise, if $P(m_1, m_2) = 0$ the output should reveal *no information* about any of the messages m_1, m_2 or m_3 — even to an adversary that knows the secret decryption key sk.

H.1 Conditional Encryption Syntax, Correctness and Error Detection

Definition 6 (Algorithms) *[AB24] A conditional encryption scheme* Π *for a binary predicate* $P : (\mathcal{M} \cup \{\perp\}) \times \mathcal{M} \to \{0,1\}$ *consists of four main algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$ *which are described as follows:*

- $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda; r)$*: takes as input the security parameter* λ *and random coins* $r \leftarrow_R \{0,1\}^{p(\lambda)}$ *and generates a secret key* $sk \in \mathcal{SK}$ *and the corresponding public key* $pk \in \mathcal{PK}$ *for our conditional encryption scheme.*
- $(b = 0, c) = \mathsf{Enc}_{pk}(m_1; r)$*: takes as input a plaintext message* $m_1 \in \mathcal{M}$ *the public key* pk *random nonce* $r \leftarrow_R \{0,1\}^{p(\lambda)}$ *and outputs a ciphertext* $(b, c) \in \{0\} \times \mathcal{C}$ *encrypting* m_1*. The flag* $b = 0$ *indicates that* c *is output of the regular encryption scheme* Enc_{pk}*.*
- $(b = 1, \tilde{c}) = \mathsf{CEnc}_{pk}\left((0, c_{m_1}), m_2, m_3; r\right)$*: This conditional encryption algorithm takes as input a public key* pk*, a ciphertext* $(0, c_{m_1})$ *with* $c_{m_1} \in \mathcal{C}$ *corresponding to an unknown message* $m_1 \in \mathcal{M}$*, a control message* m_2*, a payload message* $m_3 \in \mathcal{M}$*, and random nonce* $r \leftarrow_R \{0,1\}^{p(\lambda)}$ *and outputs a ciphertext* $(b, \tilde{c}) \in \{1\} \times \mathcal{C}$*. The flag* $b = 1$ *indicates that this ciphertext is the output of the conditional encryption* CEnc *algorithm. Note: When the control message and the payload message are the same* $m_2 = m_3$

we will sometimes write $\mathsf{CEnc}_{pk}(c_{m_1}, m_2; r)$ *instead of* $\mathsf{CEnc}_{pk}(c_{m_1}, m_2, m_2; r)$*. If the input ciphertext takes the form* $(b = 1, c_m)$ *then* $\mathsf{CEnc}_{pk}((1, c_{m_1}), m_2, m_3; r) = \perp$ *i.e.,* $(b = 0, c_{m_1})$ *must be the output of the regular encryption scheme.*

- $\{m, \perp\} = \mathsf{Dec}_{sk}(c)$*: takes as input a ciphertext* $c \in \{0,1\} \times \mathcal{C}$ *and the secret key* sk *and outputs a message* $m \in \mathcal{M}$ *or* $\perp$ *(indicating failure).*

We require that for any valid pair (sk, pk) produced that $\Pr[\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m)) = m] = 1$ i.e., perfect correctness for ciphertexts output by the regular encryption algorithm. For correctness of the conditional encryption algorithm we want to ensure that $\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(c', m_2, m_3)) = m_3$ whenever $\exists r', m_1$ s.t. $c' = \mathsf{Enc}_{pk}(m_1; r')$ and $P(m_1, m_2) = 1$. Intuitively, we can extract our intended payload m_3 if and only if $P(m_1, m_2) = 1$. For conditional encryption we relax our requirement of perfect correctness and instead require that $\Pr[\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(c', m_2, m_3)) = m_3] \geq 1 - \epsilon(\lambda)$ for a negligible function $\epsilon(\cdot)$ whenever $\exists r', m_1$ s.t. $c' = \mathsf{Enc}_{pk}(m_1; r')$ and $P(m_1, m_2) = 1$. We stress that the correctness condition only holds when when $c' \leftarrow \mathsf{Enc}_{pk}(m_1)$ was the output of the regular encryption algorithm i.e., if $c' = \mathsf{CEnc}_{pk}(c'', m_2, m_3)$ we make no guarantees. In fact, if $c = (1, \tilde{c})$ was generated by conditional encryption, then the algorithm $\mathsf{CEnc}_{pk}(c, m_2, m_3)$ can output $\perp$ as is the case in all of our constructions.

Definition 7 (Correctness) *[AB24] We say that* Π *is* $1 - \epsilon(\cdot)$*-correct if the following conditions hold:*

- ***Regular encryption correctness.*** $\forall r_1, r_2 \in \{0,1\}^{p(\lambda)}, m \in \mathcal{M}$ *we have* $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m; r_2)) = m$ *whenever* $\{sk, pk\} \leftarrow \mathsf{KeyGen}(1^\lambda; r_1)$*.*
- *(**Conditional encryption correctness.**)* $\forall r_1, r_2 \in_R \{0,1\}^{p(\lambda)}, m_1 \in \mathcal{M}, m_2, m_3 \in \mathcal{M}$ *such that* $P(m_1, m_2) = 1$ *we have*

$$\Pr[\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(\mathsf{Enc}_{pk}(m_1; r_2), m_2, m_3; r_3)) = m_3] \geq 1 - \epsilon(\lambda)$$

where the randomness is taken over the random coins r_3 *of* CEnc *and we fix* $\{sk, pk\} \leftarrow \mathsf{KeyGen}(1^\lambda; r_1)$*.*

If $\epsilon(\lambda) = 0$ *we simply say that* Π *is correct.*

Definition 8 (Efficiency) *[AB24] We say that the conditional encrypt scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$ *is efficient if all four algorithms run in probabilistic polynomial time in the security parameter* λ*.*

Optionally, conditional encryption Π should be error-detecting, i.e., whenever $P(m_1, m_2) = 0$, the ciphertext $c = \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ decrypts to a special symbol $\mathsf{Dec}_{sk}(c) = \perp$ (whp). If Π is error detecting, this allows us to detect which outputs of the CEnc algorithm are (in)valid.

Definition 9 (Error Detecting) *[AB24] We say that* Π *is* $1 - \epsilon(\cdot)$*-error detecting if* $\forall r_1, r_2 \in_R \{0,1\}^{p(\lambda)}, m_1 \in \mathcal{M}, m_2, m_3 \in \mathcal{M}$ *such that* $P(m_1, m_2) = 0$ *we have* $\Pr[\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(\mathsf{Enc}_{pk}(m_1; r_2), m_2, m_3; r_3)) = \perp] \geq 1 - \epsilon(\lambda)$ *where the randomness is taken over the selection of the random coins* r_3 *of* CEnc *and we fix* $\{sk, pk\} \leftarrow \mathsf{KeyGen}(1^\lambda; r_1)$*.*

We now formally review the security of a conditional encryption scheme. The security is defined based on a game between the challenger and adversary/distinguisher. In the security game, we ask the distinguisher to distinguish between a simulated ciphertext $\mathtt{Sim}(pk)$ and a conditionally encrypted ciphertext $\mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ — assume that $c_1 = \mathsf{Enc}_{pk}(m_1, r_1)$ with $P(m_1, m_2) = 0$. Clearly, the simulated ciphertext $\mathtt{Sim}(pk)$ cannot leak any information to the adversary as it is generated without knowledge of the control message m_2, the payload message m_3 or the ciphertext c_1.

H.1.1 Conditional Encryption Secrecy

We assume that the public key $(sk, pk) = \mathsf{KeyGen}(1^\lambda)$ and the ciphertext $c_1 = \mathsf{Enc}_{pk}(m_1, r_1)$ were both generated honestly. Semi-honest security is easier to achieve and the security guarantee is already sufficient for applications such as password typo vaults [CWP+17, AB24] where the keys and ciphertexts are generated by trusted parties.

Definition 10 (Conditional Encryption Secrecy) *[AB24] We say that conditional encryption scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$ *provides* $(t(\cdot), t_{\mathtt{Sim}}(\cdot), \epsilon(\cdot))$*-conditional encryption secrecy if there exists a simulator* $\mathtt{Sim}$ *running in time at most* $t_{\mathtt{Sim}}(\lambda)$ *such that for all messages* $m_1, m_2, m_3 \in \mathcal{M}$ *such that* $P(m_1, m_2) = 0$*, all* $\lambda \in \mathbb{N}$*,* $r, r_1 \in \{0,1\}^\lambda$ *and all distinguishers* $\mathcal{D}$ *running in time at most* $t(\lambda)$

$$\begin{aligned}&\Big|\Pr\left[\mathcal{D}\left(sk, pk, m_1, m_2, m_3, c_1, \mathtt{Sim}\left(pk\right)\right) = 1\right]\\&-\Pr\left[\mathcal{D}\left(sk, pk, m_1, m_2, m_3, c_1, \mathsf{CEnc}_{pk}\left(c_1, m_2, m_3\right)\right) = 1\right]\Big|\\&\leq \epsilon\left(t\left(\lambda\right), \lambda\right)\end{aligned} \tag{3}$$

where the randomness is taken over the random coins of the distinguisher and the conditional encryption algorithm CEnc*. We use* $(sk, pk) = \mathsf{KeyGen}(1^\lambda; r)$ *and* $c_1 = \mathsf{Enc}_{pk}(m_1; r_1)$ *to denote public/secret key and ciphertext computed under the apriori fixed random strings* r *and* r_1*. If* $\epsilon(\cdot) = 0$ *and* $t(\cdot) = \infty$ *then we say that* Π *has perfect conditional encryption secrecy.*

Real or Random Security. Similar to traditional public key encryption schemes, it is required that the encryption scheme is secure against any computationally bounded adversary who does not have the secret key sk. A conditional encryption scheme must also satisfy the traditional notion of Real-Or-Random (ROR) security for public-key encryption. In the ROR security game, the attacker is given access to an oracle O_b that takes as input a message m and outputs $\mathsf{Enc}_{pk}(m)$ when $b = 1$ or outputs a random unrelated ciphertext when $b = 0$. The attacker's goal is to predict the bit b, i.e. does the oracle output honest ciphertexts or random ones? Ameri and Blocki [AB24] extended the notion of real-or-random security to allow the attacker oracle access to the conditional encryption algorithm as well. In the following, we also review the Real-or-Random security based on the a security experiment/game called $\mathtt{CE}_{\mathcal{A}}^{ROR}(1^\lambda)$.

Intuitively, the attacker accesses a conditional encryption oracle that accepts a triple (c, m_2, m_3) as input (the attacker also accesses a regular encryption oracle that accepts m as input and returns its encryption or the encryption of a randomly chosen message). If $b = 0$, then the oracle simply returns the honestly computed ciphertext $c_0 = \mathsf{CEnc}_{pk}(c, m_2, m_3)$. Otherwise, the oracle picks random messages r' and r'' that are not related to the control message m_2 and the payload message m_3 and returns $c_1 = \mathsf{CEnc}_{pk}(c, r', r'')$. We remark that the attacker gets to pick the ciphertext c in addition to the messages m_2 and m_3. Thus, if the attacker cannot distinguish c_0 from c_1, the attacker cannot learn anything about the payload message m_2 or m_3 even if it carries out a malicious input c'.

RoR Experiment: In the security game, we pick a random $(sk, pk) = \mathsf{KeyGen}(1^\lambda; r)$ and the attacker tries to distinguish the encryption oracles $\mathsf{Enc}_{pk}(\cdot)$ and $\mathsf{CEnc}_{pk}(\cdot, \cdot)$ from random encryption oracles. More precisely, we consider the following experiment $\mathtt{CE}_{\mathcal{A}}^{ROR}(1^\lambda)$: (1) The challenger $\mathcal{C}$ picks a random bit $b \in \{0,1\}$ and generates a random public/secret key pair $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$; (2) The challenger sends pk to the attacker $\mathcal{A}(1^\lambda)$; (3) Whenever the attacker submits a message m to the encryption oracle the challenger sets $m_0 = m$ and picks a random message m_1 and sends back $\mathsf{Enc}_{pk}(m_b)$. (4) Whenever the attacker $\mathcal{A}$ submits a pair (c, m, m') to the conditional encryption oracle the challenger sets $c_0 = \mathsf{CEnc}_{pk}(c, m, m')$ and sets $c_1 = \mathsf{CEnc}_{pk}(c, r, r')$ for a random messages r, r' and sends back c_b. (5) The game ends when $\mathcal{A}$ outputs a bit b' and the output of the experiment is $\mathtt{CE}_{\mathcal{A}}^{ROR}(1^\lambda) = 1$ if and only if $b = b'$.

Definition 11 (Real or Random Security) *We say that a conditional encryption scheme* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$ *is* $(t(\cdot), q(\cdot), \epsilon(\cdot))$*-secure if for all attackers* $\mathcal{A}$ *running in time* $t(\lambda)$ *and making at most* $q(\lambda)$ *oracle queries in total we have* $\Pr\left[\mathtt{CE}_{\mathcal{A}}^{ROR}\left(1^\lambda\right) = 1\right] \leq \frac{1}{2} + \epsilon\left(t\left(\lambda\right), q\left(\lambda\right), \lambda\right)$.

H.1.2 Review: Equality Predicate Construction.

We start by quick review of Ameri and Blocki's [AB24] semi-honest conditional scheme for the equality predicate $P_=(m_1, m_2) = 1$ if and only if $m_1 = m_2$. We review this construction as a warm-up because it it a key building block in the Hamming Distance constructions. Note that an honest Pallier ciphertext $c = \mathsf{P.Enc}_{pk=N}(m_1; r)$ for the message $m_1 \in \mathbb{Z}_N$ takes the form $c = (1+N)^{m_1} r^N \mod N^2$ where $r \in \mathbb{Z}_N^*$ is

a random nonce. The conditional encryption algorithm $\mathsf{CEnc}_{pk}(c, m_2, m_3)$ returns $(1, c')$ where the flag bit 1 simply indicates that this is a conditional ciphertext and $c' = (1+N)^{-Rm_2+m_3} c^R r'^N \bmod N^2$ and the values $R \in_R \mathbb{Z}_N$ and $r' \in \mathbb{Z}_N^*$ are selected randomly. Intuitively, if $c = (1+N)^{m_1} r^N \bmod N^2$ then we can rewrite $c' = (1+N)^{R(m_1-m_2)+m_3}(rr')^N \bmod N^2$. If $m_1 = m_2$, then $c' = (1+N)^{m_3}(rr')^N \bmod N^2$ is a valid Pallier Encryption of m_3; otherwise c' can be viewed as a uniformly random sample from the ciphertext space $\mathbb{Z}_{N^2}^*$ as long as $|m_1 - m_2| < \min\{p, q\}$ and $\gcd(\phi(N) = (p-1)(q-1), N) = 1$. An honestly generated public key $N = pq$ will satisfy the properties that $\gcd((p-1)(q-1), N) = 1$ and $\min\{p, q\} \gg |\Sigma|^n$. In particular, for any messages $m_1 \leq |\Sigma|^n$ and $m_2 \leq |\Sigma|^n$ in our message space, we will have $|m_1 - m_2| \leq |\Sigma|^n < \min\{p, q\}$. Thus, to prove conditional encryption secrecy, the simulator $\mathtt{Sim}(N)$ can simply output a uniformly random sample from $\mathbb{Z}_{N^2}^*$.

H.1.3 Review of Hamming Distance Construction [AB24]

The Hamming Distance construction of [AB24] utilized Shamir Secret Sharing, Authenticated Encryption and their (semi-honest) conditional encryption scheme for the equality predicate. The regular encryption algorithm $\mathsf{Enc}_{pk}(m)$ takes as input a message $m \in \Sigma^n$ splits m into n individual characters $m[1], \ldots, m[n]$ and encrypts them one by one to obtain the output ciphertext $c = (c[1], \ldots, c[n])$. Here, $c[i] = (N+1)^{m[i]} r_i^N \bmod N^2$ for a random nonce $r_i \in \mathbb{Z}_N^*$. Given c, m_2 and m_3 as input, the conditional encryption algorithm CEnc_{pk} first picks a random symmetric key $K \in \{0,1\}^\lambda$ and encrypts the payload message m_3 with the secret key $K \in \{0,1\}^\lambda$, i.e., $c_{AE} = \mathsf{AuthEnc}_K(m_3)$. The algorithm then uses Shamir Secret Sharing to generate n shares $[\![s]\!]_1, \ldots, [\![s]\!]_n \in \mathbb{Z}_{2^\lambda}$ of the secret key K. Each share $[\![s]\!]_i$ is then encoded as a random integer $\hat{s}_i \in \mathbb{Z}_N$ subject to the constraint that $\hat{s}_i = [\![s]\!]_i \bmod 2^\lambda$. Finally, for each character i we generate a conditional ciphertext $c'[i] = (N+1)^{-Rm_2[i]+\hat{s}_i} c[i]^R r_i'^N \bmod N^2$. The output is $(1, c' = (c'[1], \ldots, c'[n]), c_{AE})$. Intuitively, if $m[i] = m_2[i]$ then we can recover the correct share $[\![s]\!]_i$ from $c'[i]$. If the Hamming Distance is at most ℓ then we will recover $n - \ell$ shares which is enough to recover K and decrypt c_{AE}. However, because we don't know a prior which shares are (in)valid the conditional decryption algorithm will have to enumerate over all $\Omega\left(\binom{n}{\ell}\right)$ subsets of $n - \ell$ shares. The proof of conditional encryption secrecy crucially relied on the observation that any subset of $n - \ell - 1$ Shamir Secret Shares can be viewed as independent uniformly random samples from our finite field of size 2^λ. The proof also critically relied on the observation that the random encoding $\hat{s}_i$ of a random share $[\![s]\!]_i$ is statistically indistinguishable from a random sample from $\mathbb{Z}_N$. Taken together these properties ensure that if the predicate does not hold $\mathtt{Ham}(m, m_2) > \ell$, that, even if the strings match at character i ($m[i] = m_2[i]$), we still cannot distinguish $c[i]$ from a uniformly random ciphertext in $\mathbb{Z}_{N^2}^*$. Thus, the simulator can simulate each $c[i]$ by simply picking a uniformly random ciphertext in $\mathbb{Z}_{N^2}^*$. For completeness we include a full description of the Hamming Distance construction from [AB24] in **Construction** 3 included in Appendix I.

H.2 Efficient Scheme for Arbitrary Hamming Distance

H.2.1 High Level Overview

Our modified construction follows a similar approach as Ameri and Blocki [AB24]. The key difference is that the conditional encryption algorithm uses $(n - \ell, n, \epsilon(\lambda))$-Random Robust Secret Sharing to generate n shares $[\![s]\!]_1, \ldots, [\![s]\!]_n \in \mathbb{Z}_{2^\lambda}$ of the secret key K. Because the RRSS share $[\![s]\!]_i$ can be a bit larger it may be necessary to split the string $[\![s]\!]_i$ into smaller pieces $s_{i,1}, \ldots, s_{i,\mathsf{d_{Pack}}}$ each of which can be encoded as a random integer $\hat{s}_{i,j}$ in $\mathbb{Z}_N$ subject to the constraint that $\hat{s}_{i,j} = s_{i,j} \bmod N$. In particular, for each character i we will generate $\mathsf{d_{Pack}}$ conditional ciphertexts $c'[i,j] = (N+1)^{-R_{i,j} m_2[i] + \hat{s}_{i,j}} c[i]^{R_{i,j}} r_{i,j}'^N \bmod N^2$. The final output includes $c'[i,j]$ for each $i \leq n$ and $j \leq \mathsf{d_{Pack}}$. Intuitively, if $m[i] = m_2[i]$ then for each $j \leq \mathsf{d_{Pack}}$ the ciphertext $c'[i,j]$ will decrypt to $\hat{s}_{i,j}$ allowing us to reconstruct the original share $[\![s]\!]_i$. On the other hand if $m[i] \neq m_2[i]$ then we will essentially end up constructing a uniformly random share $[\![s']\!]_i$. Thus, if $\mathtt{Ham}(m, m_2) \leq \ell$ (the Hamming Distance between m and m_2 is at most ℓ) then we will recover $n - \ell$ correct shares with ℓ random shares mixed in. We can then utilize the $\mathsf{Identify}$ algorithm from our RRSS construction to find $n - \ell$ correct shares and recover the secret key K. Thus, the simulator can simulate a conditional ciphertext by simply picking $c'[i,j] \in_R \mathbb{Z}_{N^2}^*$ for each $i \leq n$ and $j \leq \mathsf{d_{Pack}}$. On the other hand if $\mathtt{Ham}(m, m_2) > \ell$ (the predicate

does not hold) we can appeal to $n-\ell-1$ perfect privacy to argue the uncorrupted shares are uniformly distributed.

H.2.2 Construction

To obtain a more efficient construction for arbitrary ℓ, we use $(n,t,\epsilon(\lambda))$-RRSS scheme $\Pi_{RRSS} = (\mathsf{Setup}, \mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$ described in **Construction 1**. We observed that we can reduce the decryption overhead, when we use the RRSS due to $\Pi_{RRSS}.\mathsf{Identify}$ algorithm. In this section, we formally describe our generic construction of conditional encryption for *arbitrary* Hamming distance with efficient decryption algorithm which uses our Π_{RRSS}, Paillier Encryption and authenticated encryption as the constructive building blocks. In what follows, we present a detailed description of our conditional encryption scheme $\Pi_{\mathtt{Ham},\ell} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$, designed for the Hamming distance predicate $P_{\mathtt{Ham},\ell}$ with arbitrary ℓ.

Because the shares of a RRSS scheme can be a bit longer it is useful to split each share $[\![s]\!]_i$ into smaller pieces and then randomly encode each piece as an integer in $\mathbb{Z}_N$. For simplicity we will assume that each share is a uniformly random m-bit string — for our particular construction we have $m = 6n\lambda_1 + \lambda$. We can split the string $[\![s]\!]_i \in \{0,1\}^m$ into $\mathsf{d_{Pack}}$ substrings $s_1, \ldots, s_{\mathsf{d_{Pack}}}$ such that the first $\mathsf{d_{Pack}} - 1$ substrings have length $|s_i| \leq \left\lceil \frac{m}{\mathsf{d_{Pack}}} \right\rceil$ and the final substring $s_{\mathsf{d_{Pack}}} \in \{0,1\}^r$ has length $r \leq \frac{m}{\mathsf{d_{Pack}}}$. We also rely on a randomized encoding procedure $\mathsf{REnc}(x,a,N)$ which takes as input integers $0 \leq x < a \leq N$ and randomly picks an integer $0 \leq y < N$ subject to the constraint that $y \bmod a = x$. We will slightly abuse notation interpret each $s_i \in \{0,1\}^{\lceil \frac{m}{d} \rceil}$ as an integer between 0 and $2^{\lceil \frac{m}{d} \rceil} - 1$ and encode $\mathsf{REnc}(z_i,a,N)$ with $a = 2^{\lceil \frac{m}{d} \rceil} - 1$. Ameri and Blocki [AB24] observed that as long as $a \ll N$ and z_i is uniformly random between 0 and $a-1$ the distribution of $\mathsf{REnc}(z_i,a,N)$ will be statistically indistinguishable from the uniform distribution over $\mathbb{Z}_N$.

- **Key Generation** $(pk', sk') \leftarrow \Pi_{\ell,\mathtt{Ham}}.\mathsf{KeyGen}(1^\lambda, n, \ell)$**:** This algorithm first runs the setup algorithm $\mathsf{pp} = (\vec{v}, n, t = n-\ell, \Pi^{\lambda}_{SSS,n,t}, \Pi^{\lambda_1}_{SSS,2n,2t}) \leftarrow \Pi_{RRSS}.\mathsf{Setup}(1^\lambda, n, t)$. The algorithm returns $(pk' = (pk, \mathsf{pp}), sk' = sk)$.
- **Regular Encryption** $c \leftarrow \Pi_{\ell,\mathtt{Ham}}.\mathsf{Enc}_{pk'}(m)$**:** The algorithm $\Pi.\mathsf{Enc}_{pk'}(m)$ takes as input the message $m = (m[1], \ldots, m[n]) \in \Sigma^n$ and public key $pk = N \in pk'$ and encrypts m character by character and computes $c = (c[1], \ldots, c[n])$ in which $c[i] = \mathsf{P}.\mathsf{Enc}_{pk}(m[i]; r_i) = (1+N)^{m[i]} r_i^N \bmod N^2$. The final regular encryption output is $(b = 0, c[1], \ldots c[n])$.
- **Conditional Encryption** $c \leftarrow \Pi.\mathsf{CEnc}_{pk'}(c, m_2, m_3)$**:** The algorithm $\Pi_{\ell,\mathtt{Ham}}.\mathsf{CEnc}_{pk'}(c_1, m_2, m_3)$ takes as input the public key $pk' = (pk, \mathsf{pp})$, a ciphertext $c_1 = (b = 0, c_1[1], \ldots, c_1[n])$ corresponding to some unknown message $m_1 = (m_1[1], \ldots, m_1[n])$, a control message $m_2 = (m_2[1], \ldots m_2[n]) \in \Sigma^n$ and a payload message m_3. The algorithm performs the following steps to generate the ciphertext.
  (1) Randomly pick the key $K \in \{0,1\}^\lambda$ and compute $c_{AE} = \mathsf{AuthEnc}_K(m_3)$.
  (2) Compute n shares $([\![s]\!]_1, \ldots, [\![s]\!]_n) \leftarrow \Pi_{RRSS}.\mathsf{ShareGen}(K, \mathsf{pp} \in pk')$.
  (3) $\forall 1 \leq i \leq n$:
    - Partition each $[\![s]\!]_i \in \{0,1\}^m$ into $\mathsf{d_{Pack}}$ strings $s_{1,1}, \ldots, s_{1,\mathsf{d_{Pack}}}$ such that $|s_{i,j}| = \lceil m/\mathsf{d_{Pack}} \rceil$ for $j < \mathsf{d_{Pack}}$ and $|s_{1,\mathsf{d_{Pack}}}| = m - \sum_{j=1}^{\mathsf{d_{Pack}}-1} |s_{i,j}|$.
    - Compute $\hat{s}_{i,j} = \mathsf{REnc}(s_{i,j}, 2^{\lceil m/\mathsf{d_{Pack}} \rceil}, N)$ for all $j < \mathsf{d_{Pack}}$ and $\hat{s}_{i,\mathsf{d_{Pack}}} = \mathsf{REnc}(s_{i,\mathsf{d_{Pack}}}, 2^{m - \mathsf{d_{Pack}}\lceil m/\mathsf{d_{Pack}} \rceil + \lceil m/\mathsf{d_{Pack}} \rceil}, N)$.
    - For each $j \leq \mathsf{d_{Pack}}$ compute $c_i^j = (N+1)^{R_{i,j} m_2[i] + \hat{s}_{i,j}} c[i]^{R_{i,j}} r_{i,j}^N \bmod N^2$. Note that if $m_2[i] = m[i]$ (the two strings match at character i) then c_i^j will be a valid Pallier encryption of $\hat{s}_{i,j}$. Otherwise, c_i^j will be a uniformly random Pallier ciphertext in $\mathbb{Z}^*_{N^2}$.
    - Set $c'[i] = (c_i^1, \ldots, c_i^{\mathsf{d_{Pack}}})$.

(4) Set the final ciphertext as $(b = 1, c, c_{AE})$ in which $c = (c'[1], \ldots, c'[n])$.

- **Decryption** $m = \Pi.\mathsf{Dec}_{sk}(c)$**:**
  - **Regular ciphertext**: Given a ciphertext $(b = 0, c[1], \ldots, c[n])$ with $b = 0$ the decryption algorithm will simply decrypt character by character to recover $m = (m[1], \ldots, m[n])$ where $m[i] = \mathtt{P}.\mathsf{Dec}_{sk}(c[i])$.
  - **Conditionally generated ciphertext**: Given a conditionally encrypted ciphertext $(b = 1, c[1], \ldots, c[n], c_{AE})$ this algorithm executes the following steps:
    (1) For all $\forall i \leq n$: Parse $c[i] = (c_i^1, \ldots, c_i^{\mathsf{d_{Pack}}})$
      - $\forall j \leq \mathsf{d_{Pack}}$: compute $\hat{s}'_{i,j} = \mathtt{P}.\mathsf{Dec}_{sk}(c_i^j)$
      - $\forall j < \mathsf{d_{Pack}}$: compute $s'_{i,j} = \hat{s}'_{i,j} \bmod 2^{\lceil m/\mathsf{d_{Pack}} \rceil}$.
      - compute $s'_{i,\mathsf{d_{Pack}}} = \hat{s}'_{i,\mathsf{d_{Pack}}} \bmod 2^{m - \mathsf{d_{Pack}}\lceil m/\mathsf{d_{Pack}} \rceil + \lceil m/\mathsf{d_{Pack}} \rceil}$.
      - Interpret each $s'_{i,j}$ as a bit string and set $[\![s']\!]_i = s'_{i,1} \circ \ldots \circ s'_{i,\mathsf{d_{Pack}}}$.

    (2) Set $\vec{s}' = ([\![s']\!]_1, \ldots, [\![s']\!]_n)$ and compute $K' := \Pi_{RRSS}.\mathsf{SecretRecover}(\vec{s}', \mathsf{pp})$.

    (3) Return $\{m, \perp\} := \mathsf{AuthDec}_{K'}(c_{AE})$.

In the remainder of the paper, we refer to **Construction 2** as our efficient conditional encryption construction for arbitrary Hamming-distance predicates, featuring an efficient RRSS-based decryption algorithm and presented in a separate algorithmic environment for clarity and convenience.

We highlight that the construction is very similar to Ameri and Blocki's construction for Hamming distance [AB24], however, we replace the Shamir secret sharing with our RRSS. Intuitively, the decryption algorithm invokes the identification algorithm $\Pi_{\mathsf{RRSS}}.\mathsf{Identify}$ (is called inside $\Pi_{\mathsf{RRSS}}.\mathsf{SecretRecover}$) to find the location of valid shares which are places that the characters of the control message m_2 and the unknown message m_1 are matching. If we have enough matching shares, then we can apply the correctness of RRSS to argue that we we can recover the valid shares to reconstruct the authenticated encryption K (is used to decrypt the payload m_3). If the Hamming distance is more than the threshold value, then decryption returns $\perp$ since we cannot find enough valid shares.

We argue that one trivial way (if we use traditional Shamir Secret Sharing as in [AB24]) is to check all possible $\binom{n}{n-\ell}$ possible subsets which may not be efficient. That is why we use our Π_{RRSS} is $(k = n - \ell, 2^{-n\lambda})$-random robust share identification (Definition 1) which inherently finds the subset of $k = n - \ell$ valid shares way faster ($O(n^\omega)$: complexity of solving a system of linear equations of size $O(n) \times O(n)$) than the brute force.

H.3 Proving Conditional Encryption Secrecy for Construction 2

Theorem 14 (Conditional Encryption Secrecy of Construction 2) *Assume that our Authenticated encryption scheme* $\Pi_{AE} = (\mathsf{AuthEnc}, \mathsf{AuthDec})$ *is* $(t_{AE}, \epsilon_{AE}(t_{AE}, \lambda))$*-secure for any security parameter* λ *and any running time parameter* t_{AE}*. Also, let* $\Pi_{RRSS} = (\mathsf{ShareGen}_{2^\lambda}, \mathsf{SecretRecover}_{2^\lambda}, \mathsf{Identify}_{2^\lambda})$ *be a* $(n, k = n - \ell, \epsilon(\lambda))$*-Random Robust secret sharing scheme. Then for any* t *and any security parameter* λ ***Construction 2*** *provides* $(t, t_{\mathtt{Sim}}, \epsilon(t, \lambda))$*-conditional encryption secrecy with* $\epsilon(t, \lambda) \leq \epsilon_{AE}(t', \lambda) + 2^{-\lambda}$*,* $t = t_{AE}(\lambda)$ *and* $t_{\mathtt{Sim}} = n\mathsf{d_{Pack}} \cdot t_{P.\mathsf{Enc}} + \mathtt{poly}(\lambda)$*.*

Proof of Theorem 14: For this proof, we use hybrid argument and we define a sequence of indistinguishable hybrids. In the first hybrid, the distinguisher is given the output of conditional encryption ciphertext, and in the last hybrid, the distinguisher is given the output of our simulator described in Figure 10. Indistinguishability of these hybrids implies the indistinguishability of the conditional ciphertext and the simulator $\mathtt{Sim}(pk)$, that is, satisfying conditional encryption secrecy in the malicious setting.

Construction 2

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda, n, \ell)$:

1. run $(\mathtt{P}.sk, \mathtt{P}.pk = (N)) \leftarrow \mathtt{P}.\mathsf{KeyGen}(1^\lambda; r)$.
2. set $\mathtt{SS} = \Pi_{RRSS}.(\mathsf{ShareGen}, \mathsf{SecretRecover}, \mathsf{Identify})$ over finite field $\mathbb{F}_{2^\lambda}$.
3. set $t = n - \ell$, compute $\mathsf{pp} = (\vec{v}, n, t, \Pi^\lambda_{SSS,n,t}, \Pi^{\lambda_1}_{SSS,2n,2t}) \leftarrow \mathtt{SS}.\mathsf{Setup}(1^\lambda, n, t)$
4. return $\left(pk' = (\mathtt{P}.pk = N, \mathsf{pp}), sk' = \mathtt{P}.sk\right)$.

$c \leftarrow \mathsf{Enc}_{pk}(m; r)$:

1. parse $m = (m[1], \ldots, m[n]), r = (r_1, \ldots, r_n) \in (\mathbb{Z}^*_N)^n$.
2. $\forall 1 \le i \le n$:
   - compute $\hat{m}[i] = \mathsf{ToInt}(m[i])$
   - compute $c_i = (1+N)^{\hat{m}[i]} {r_i}^N \mod N^2$
3. output $c = (b = 0, c_1, \ldots, c_n)$.

$c \leftarrow \mathsf{CEnc}_{pk}(c_m, m_2, m_3; r)$:

1. parse $m_2 = (m_2[1], \ldots, m_2[n]) \in \Sigma^n$, $r = (r_1, \ldots, r_n) \in (\mathbb{Z}^*_N)^n$, $c_m = (b, c_1, \cdots, c_n)$, $pk = (\mathtt{P}.pk = N, \mathsf{pp}, \mathsf{d_{Pack}}, \lambda, \lambda_1)$;
2. if $b = 1$, output $\perp$.
4. $\forall i \in [n]$: compute $\hat{m}_2[i] = \mathsf{ToInt}(m_2[i])$;
5. sample $K \in_R \{0,1\}^\lambda$, compute $c_{AE} = \mathsf{Auth.Enc}_K(m_3)$;
6. $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \mathtt{SS}.\mathsf{ShareGen}(K, \mathsf{pp})$;
7. $\forall 1 \le i \le n$:
   - Partition each $\llbracket s \rrbracket_i \in \{0,1\}^m$ into $\mathsf{d_{Pack}}$ strings $s_{1,1}, \ldots, s_{1,\mathsf{d_{Pack}}}$ such that $|s_{i,j}| = \lceil m/\mathsf{d_{Pack}} \rceil$ for $j < \mathsf{d_{Pack}}$ and $|s_{1,\mathsf{d_{Pack}}}| = m - \sum_{j=1}^{\mathsf{d_{Pack}}} |s_{i,j}|$.
   - Compute $\hat{s}_{i,j} = \mathsf{REnc}(s_{i,j}, 2^{\lceil m/\mathsf{d_{Pack}} \rceil}, N)$ for all $j < \mathsf{d_{Pack}}$ and $\hat{s}_{i,j} = \mathsf{REnc}(s_{i,j}, 2^{m - \mathsf{d_{Pack}}\lceil m/\mathsf{d_{Pack}} \rceil + \lceil m/\mathsf{d_{Pack}} \rceil}, N)$.
   - For each $j \le \mathsf{d_{Pack}}$ compute $c_i^j = (N+1)^{R_{i,j} m_2[i] + \hat{s}_{i,j}} c[i]^{R_{i,j}} r_{i,j}^N \mod N^2$. Note that if $m_2[i] = m[i]$ (the two strings match at character i) then c_i^j will be a valid Pallier encryption of $\hat{s}_{i,j}$. Otherwise, c_i^j will be a uniformly random Pallier ciphertext in $\mathbb{Z}^*_{N^2}$.
   - Set $c'[i] = (c_i^1, \ldots, c_i^{\mathsf{d_{Pack}}})$.
8. Set the final ciphertext as $(b = 1, c, c_{AE})$ in which $c = \left(c'[1], \ldots, c'[n]\right)$.

$\{m, \perp\} = \mathsf{CDec}_{sk}((b, c))$:

1 if $b = 1$ // The ciphertext is extracted from $\mathsf{CEnc}(\cdot)$

1.i For all $\forall i \le n$: Parse $c[i] = (c_i^1, \ldots, c_i^{\mathsf{d_{Pack}}})$
   - $\forall j \le \mathsf{d_{Pack}}$: compute $\hat{s}'_{i,j} = \mathtt{P}.\mathsf{Dec}_{sk}(c_i^j)$
   - $\forall j < \mathsf{d_{Pack}}$: compute $s'_{i,j} = \hat{s}'_{i,j} \mod 2^{\lceil m/\mathsf{d_{Pack}} \rceil}$.
   - compute $s'_{i,\mathsf{d_{Pack}}} = \hat{s}'_{i,\mathsf{d_{Pack}}} \mod 2^{m - \mathsf{d_{Pack}}\lceil m/\mathsf{d_{Pack}} \rceil + \lceil m/\mathsf{d_{Pack}} \rceil}$.
   - Interpret each $s'_{i,j}$ as a bit string and set $\llbracket s' \rrbracket_i = s'_{i,1} \circ \ldots \circ s'_{i,\mathsf{d_{Pack}}}$.

1.ii Set $\vec{s}' = (\llbracket s' \rrbracket_1, \ldots, \llbracket s' \rrbracket_n)$ and compute $K' := \Pi_{RRSS}.\mathsf{SecretRecover}\left(\vec{s}', \mathsf{pp}\right)$.

1.iii Return $\{m, \perp\} := \mathsf{AuthDec}_{K'}(c_{AE})$.

2 if $b = 0$ // The ciphertext is extracted from $\mathsf{Enc}(\cdot)$
   - parse $c = (c_1, \cdots, c_n)$, AND set $m = \perp$
   - for $1 \le i \le n$, compute $\hat{m}_i = \mathtt{P}.\mathsf{Dec}_{sk}(c_i)$
   - return $m = \mathsf{ToInt}^{-1}(\hat{m}_1) \| \ldots \| \mathsf{ToInt}^{-1}(\hat{m}_n)$

Figure 9: Concrete construction of conditional encryption over binary predicate $P_{\ell,\mathtt{Ham}}$.

Design of simulator $\mathtt{Sim}(pk)$

1. Sample, $r''_1, \ldots, r''_{\mathsf{d_{Pack}} \times n} \in_R {\mathbb{Z}^*_N}^n, R''_1, \ldots, R''_n \in_R {\mathbb{Z}_N}^{\mathsf{d_{Pack}} \times n}$ uniformly at random
2. For all $1 \le i \le n$ compute $\tilde{c}'_i = \mathtt{P}.\mathsf{Enc}_{pk}(R''_i; r''_i)$
3. Pick $R_K \in_R \{0,1\}^{l(\lambda)}$ uniformly at random and set $c'_{AE} = R_K$. //$l(\lambda)$ represents the ciphertext size of our authenticated encryption.
4. Output $\tilde{c}' = (1, \tilde{c}'_1, \cdots, \tilde{c}'_{n\mathsf{d_{Pack}}}, c'_{AE})$.

Figure 10: The simulator $\mathtt{Sim}$ for **Construction 2**

- **Hybrid 0**: In this hybrid the distinguisher $\mathcal{D}$ is given $(sk, pk, m_2, m_3, c_1, (1, \mathsf{d_{Pack}}, \tilde{c}))$ in which $\tilde{c} = (\tilde{c}_1, \ldots, \tilde{c}_n, c_{AE}) \leftarrow \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$.
- **Hybrid 1**: Let $T_1 = \{j : m_2[j] \neq m_1[j]\}$. We define **Hybrid 1** similar to **Hybird 0**, except for all $j \in T_1$ we replace

$$\tilde{c}[j] = (c[j,i] = \mathsf{P.Enc}_{pk}(R_{j,i}(m_1[j] - m_2[j]) + \mathsf{REnc}(p_{j,i}, a_i, N)))_{i=1}^{\mathsf{d_{Pack}}}$$

  with $\tilde{c}[j] = \left(\mathsf{P.Enc}_{pk}(R'_{j,i})\right)_{i=1}^{\mathsf{d_{Pack}}}$ where $R'_{j,i} \in_R \mathbb{Z}_N$ are fresh and uniform random values chosen from $\mathbb{Z}_N$ and $a_i = 2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}$ for all $1 \leq i \leq \mathsf{d_{Pack}} - 1$ and $a_{\mathsf{d_{Pack}}} = 2^{m - \mathsf{d_{Pack}} \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil + \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}$.
- **Hybrid 2** Let $T_2 = [n] \setminus (T_1)$ be the remaining indices. We define **Hybrid 2** similar to **Hybird 1**, except for all $j \in T_2$ we replace

$$\tilde{c}[j] = (c[j,i] = \mathsf{P.Enc}_{pk}(R_{j,i}(m_1[j] - m_2[j]) + \mathsf{REnc}(p_{j,i}, a_i, N)))_{i=1}^{\mathsf{d_{Pack}}}$$

  with

$$\tilde{c}[j] = (c[j,i] = \mathsf{P.Enc}_{pk}(R_{j,i}(m_1[j] - m_2[j]) + \mathsf{REnc}(r_{j,i}, a_i, N)))_{i=1}^{\mathsf{d_{Pack}}}$$

  where $r_{j,i} \in_R [a_i]$ are fresh and uniform random values chosen from $[a_i]$ and $a_i = 2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}$ for all $1 \leq i \leq \mathsf{d_{Pack}} - 1$ and $a_{\mathsf{d_{Pack}}} = 2^{m - \mathsf{d_{Pack}} \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil + \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}$.
- **Hybrid 3** We define **Hybrid 3** similar to **Hybird 2**, except for all $j \in T_2$ we replace

$$\tilde{c}[j] = (c[j,i] = \mathsf{P.Enc}_{pk}(R_{j,i}(m_1[j] - m_2[j]) + \mathsf{REnc}(r_{j,i}, a_i, N)))_{i=1}^{\mathsf{d_{Pack}}}$$

  with

$$\tilde{c}[j] = \left(c[j,i] = \mathsf{P.Enc}_{pk}\left(R_{j,i}(m_1[j] - m_2[j]) + \hat{R}_{j,i}\right)\right)_{i=1}^{\mathsf{d_{Pack}}}$$

  where $\hat{R}_{j,i} \in_R \mathbb{Z}_N$ are fresh and uniform random values chosen from $\mathbb{Z}_N$ and $a_i = 2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}$ for all $1 \leq i \leq \mathsf{d_{Pack}} - 1$ and $a_{\mathsf{d_{Pack}}} = 2^{m - \mathsf{d_{Pack}} \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil + \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}$.
- **Hybrid 4** We define **Hybrid 4** similar to **Hybird 3**, except for all $j \in T_2$ we replace

$$\tilde{c}[j] = \left(c[j,i] = \mathsf{P.Enc}_{pk}\left(R_{j,i}(m_1[j] - m_2[j]) + \hat{R}_{i,j}\right)\right)_{i=1}^{\mathsf{d_{Pack}}}$$

  with $\tilde{c}[j] = \left(\mathsf{P.Enc}_{pk}(R'_{j,i})\right)_{i=1}^{\mathsf{d_{Pack}}}$ where $R'_{j,i} \in_R \mathbb{Z}_N$ are fresh and uniform random values chosen from $\mathbb{Z}_N$.

  Hybrid 5: This hybrid is exactly the same as the previous hybrid except we replace c_{AE} with $c'_{AE} \in_R \{0,1\}^{l(\lambda)}$ a $l(\lambda)$-bit string chosen uniformly at random. We note that $l(\lambda)$ is a polynomial over the security parameter λ which represents the ciphertext size of authenticated encryption.
- **Hybrid 6**: We replace the ciphertext of the conditional encryption with the output of the simulator `Sim` described in Figure 10.

Now, we will show that all these hybrids are indistinguishable. We define $\mathcal{D}^i = 1$ to denote that the event that the distinguisher output 1 in hybrid i.

- **Hybrid 0 ≡ Hybrid 1**: These hybrids are equivalent and we have $\Pr[\mathcal{D}^{H_0} = 1] = \Pr[\mathcal{D}^{H_1} = 1]$. The argument is similar to the security proof for the equality test, as well as the first hybrid in the proof of Hamming distance in [AB24]. Intuitively, since these are the location of non-matching characters, then the condition encryption decrypts to a uniformly chosen random number and we can replace all the corresponding ciphertexts with uniformly chosen ciphertext.

- **Hybrid 1 ≡ Hybrid 2**: We are replacing each

  In Hybrid 1 we information theoretically eliminated information about any share $s]\!]_i$ with $j \in T_1$. Since $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 0$, we have $|T_1| > \ell$ which implies that the set T_2 of valid shares has size $|T_2| < n - \ell$. Let $T_2 = \{i_1, \ldots, i_t\}$ with $t < n - \ell$. $n - \ell - 1$-perfect privacy of our RRSS (**Construction 1**) guarantees that the shares $\left([\![s]\!]_{i_1}, \ldots, [\![s]\!]_{i_t}\right)$ are uniformly random elements and we can replace $[\![s]\!]_j$ with the random element $s_{j,r}$. This is equivalent to picking uniformly binary string $\hat{s}_{j,r} \in \{0,1\}^m$ and divide it in $\mathsf{d_{Pack}}$ smaller parts. We then compute its equivalent integers in either $\mathbb{Z}_{a_i}$, that is $(p_{j,i,r})_{\forall i \in [\mathsf{d_{Pack}}]} \in \mathbb{Z}^{\mathsf{d_{Pack}}-1}_{2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}} \times \mathbb{Z}_{a_{\mathsf{d_{Pack}}}}$. Then, $(p_{j,i,r})_{\forall i \in [\mathsf{d_{Pack}}]}$ is a vector of integers representing $\hat{s}_{j,r}$ through a one-to-one mapping of 2^m possible m-bit share $\hat{s}_{j,r}$ to $(2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil})^{\mathsf{d_{Pack}}-1} \times a_{\mathsf{d_{Pack}}} = (2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil})^{\mathsf{d_{Pack}}-1} \times 2^{m-\mathsf{d_{Pack}}\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil + \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil} = 2^m$ possible vectors of $\mathsf{d_{Pack}}$ integers. So, all the individual $\hat{p}_{i,j,r}$ integers look uniformly at random and we have $\Pr[\mathcal{D}^1 = 1] = \Pr[\mathcal{D}^2 = 1]$.

- **Statistically indistinguishability of Hybrid 2 and 3** For each $i \in T_2$ and each $i \in [\mathsf{d_{Pack}}]$ we are replacing the randomized encoding $\mathsf{REnc}(\hat{p}_{i,j,r}, a_i, N)$ with a uniformly random element in $\mathbb{Z}_N$. To bound the statistical distance we apply Theorem 15 with parameters $a \in \{2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}, 2^{m-\mathsf{d_{Pack}}\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil + \lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}\} \leq 2^{m/\mathsf{d_{Pack}}}$, $k = \lfloor \frac{N}{a} \rfloor$ and $b = N = pq$. Union bounding over all shares $i \in T_2$ and all $j \in [\mathsf{d_{Pack}}]$ the statistical distance between Hybrid 2 and 3 is upper bounded by

$$n\mathsf{d_{Pack}} \frac{2^{m/\mathsf{d_{Pack}}}}{N} \leq 2^{-\lambda}$$

  By proper selection of the values of $\mathsf{d_{Pack}}$ and N we ensure that $n\mathsf{d_{Pack}} \frac{2^{m/\mathsf{d_{Pack}}}}{N} \leq 2^{-\lambda}$ and we get the distinguishing advantage we like, i.e., $2^{-\lambda}$. So we have $\left|\Pr\left[\mathcal{D}^2 = 1\right] - \Pr\left[\mathcal{D}^3 = 1\right]\right| \leq 2^{-\lambda}$.

- **Hybrid 3≡ Hybrid 4**: These hybrids are information theoretically indistinguishable as $R_{i,j}(m_1[j] - m_2[j]) + R'_{j,i}$ is already uniformly random in $\mathbb{Z}_N$. This is because of the fact that the j-th character of the control message m_2 and unknown message m_1 are not matching, i.e., $m_2[j] \neq m_1[j]$ and we have $m_1[j] - m_2[j] \neq 0$. Since we are in the semi-hones model, we have $m_1[j] - m_2[j] < \min\{p, q\}$ and the message space is smaller than $\min\{p, q\}$. This implies that $\gcd(m_1[j] - m_2[j], N) = 1$ and therefore $R_{i,j}(m_1[j] - m_2[j])$ is uniformly random i.e., for all $x \in \mathbb{Z}_N$ we have $\Pr_R[R(m_1[j] - m_2[j]) = x] = \Pr_R[R = x(m_1[j] - m_2[j])^{-1}] = \frac{1}{N}$. Therefore, these two hybrids are information theoretically equivalent and we have $\Pr[\mathcal{D}^3 = 1] = \Pr[\mathcal{D}^4 = 1]$.

- **Indistinguishability of Hybrid 4 and 5**: By **Hybrid 4**, we have information theoretically eliminated any information about the secret key K for our authentication encryption scheme from $(\tilde{c}_1, \ldots, \tilde{c}_n)$. Thus, by AE security any adversary running in time at most $t_{AE} = t_{AE}(\lambda)$ can distinguish between c_{AE} and c'_{AE} with the advantage of at most $\epsilon_{AE}(t_{AE}, \lambda)$. We defined $t = t_{AE}$ so we have

$$\left|\Pr\left[\mathcal{D}^{H_4}\right] - \Pr\left[\mathcal{D}^{H_5} = 1\right]\right| \leq \epsilon_{AE}(t_{AE}, \lambda) \tag{4}$$

  Hybrid 5≡ Hybrid 6 Looking at the definition of our simulator in Figure 10, we can see that the conditionally encrypted ciphertext in Hybrids 5 and 6 are generated in exactly the same way. It follows that the hybrids are information-theoretically equivalent and we have

$$\Pr\left[\mathcal{D}^{H_5}\right] = \Pr\left[\mathcal{D}^{H_6}\right] \tag{5}$$

Putting everything together we have

$$\begin{aligned}
&\Big| \Pr\left[\mathcal{D}\left(sk, pk, m_2, m_3.\mathsf{CEnc}_{pk}\left(\mathsf{Enc}_{pk}\left(m_1\right), m_2, m_3\right)\right) = 1\right] \\
&- \Pr\left[\mathcal{D}\left(sk, pk, m_2, m_3, \mathtt{Sim}\left(pk\right)\right) = 1\right] \Big| \\
&= \left|\Pr\left[\mathcal{D}^{H_0}\right] - \Pr\left[\mathcal{D}^{H_6}\right]\right| \\
&\leq \sum_{i=0}^{5} \left|\Pr\left[\mathcal{D}^{H_i}\right] - \Pr\left[\mathcal{D}^{H_{i+1}}\right]\right| \\
&< \epsilon_{AE}(t', \lambda) + 2^{-\lambda} \ .
\end{aligned}$$

□

Statistical Distance for Randomized Encoding Our analysis of conditional encryption security utilizes the following observation of Ameri and Blocki [AB24]. Theorem 15 is used to analyze the statistical distance between $\mathsf{REnc}(x, a, N)$ (when x randomly chosen in $\mathbb{Z}_a$) and the uniform distribution over $\mathbb{Z}_N$.

Theorem 15 ([AB24]) *Let $b = ak + r$ where $0 \leq r < a$ is the reminder (i.e., $r = b \mod a$). Consider the uniform distributions $\mathcal{U}_b$ which outputs a random value in $\mathbb{Z}_b$ and the distribution $\mathcal{U}_{ak}$ which outputs random values between $0, \cdots, ak - 1$. Then the statistical distance between these two distributions is $\mathtt{SD}(\mathcal{U}_{ak}, \mathcal{U}_b) = \frac{r}{b} \leq \frac{1}{k+1}$.*

H.4 Proving Correctness for Construction 2

Theorem 16 (Correctness of Construction 2) *Let Π_{RRSS} be an $(n, k = n - \ell, \epsilon_{\mathsf{RRSS}}(\lambda) = 2^{-\lambda})$-Random Robust secret sharing scheme. Then **Construction 2** is a $1 - \epsilon_1(\lambda)$-correct with $\epsilon_1(\lambda) = 2^{-\lambda+1}$. The scheme is $1 - \epsilon(\lambda)$-error detecting conditional encryption scheme with $\epsilon(\lambda) = \binom{n}{\ell}\epsilon_{AE}(\lambda) + 2^{-\lambda}$. Here,*

$$\epsilon_{AE}\left(\lambda\right) \doteq \max_{m} \Pr_{K_1, K_2 \in \{0,1\}^\lambda}\left[\mathsf{Auth.Dec}_{K_1}\left(\mathsf{Auth.Enc}_{K_2}\left(m\right)\right) \neq \perp\right]$$

denotes the negligible probability that the ciphertext $c = \mathsf{Enc}_{K_2}(m)$ is still valid under an unrelated key K_1. If Π_{RRSS} achieves $(k - 1, \epsilon_2(\lambda))$-security against malicious dealers then the scheme is $1 - \epsilon_2(\lambda) - 2^{-\lambda}$ error detecting.

Proof of Theorem 16: We first note Authenticated Encryption security implies that the term $\epsilon_{AE}(\lambda)$ is negligible. Otherwise, an AE attacker could simply pick a random key K' and use $c = \mathsf{Enc}_{K'}(m)$ as an attempted forgery for the unknown secret key K!

There are two conditions in Definition 7 which need to be proved. The first condition is regular encryption correctness and the other one is conditional encryption correctness. We also need to show that the scheme is error detecting.

Correct of Regular Encryption: The observation that $\mathsf{Dec}_{sk}\left(\mathsf{Enc}_{pk}(m; r)\right) = m$ for all messages $m \in \Sigma^n$, random coins r and all (sk, pk) in the support of the honest key generation algorithm follows immediately from the correctness of Pallier encryption. We now consider correctness for the conditional encryption algorithm.

Correctness of Conditional Encryption: We show that for all messages $m_1, m_2 \in \Sigma^n$ such that $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 1$, all payload messages m_3, all $\{sk, pk\}$ in the support of our Key Generation algorithm and all random coins r_1 used to generate $c_1 = \mathsf{Enc}_{pk}(m_1; r_1)$ we have

$$\begin{aligned}
&\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c_1, m_2, m_3; r_2\right)\right) = m_3 | c_1 = \mathsf{Enc}_{pk}\left(m_1; r_1\right)\right] \\
&\quad \geq 1 - \epsilon_1(\lambda)
\end{aligned}$$

where the randomness is taken over the selection of the random coins r_2 of the conditional encryption algorithm CEnc.

Let K denote the authenticated encryption key generated when we run CEnc, let $\mathsf{d}_{\mathsf{Pack}} \in pk$, let $[\![s]\!]_1, \ldots, [\![s]\!]_n$ denote the shares $\{[\![s]\!]_i \in \{0,1\}^m\}_{\forall i \in [n]}$ of K such that are generated by our underlying random robust secret sharing. Let $\tilde{c} = (b, \tilde{c}_1, \ldots, \tilde{c}_n, C_{AE})$ denote the output of $\mathsf{CEnc}_{pk}(c_1, m_2, m_3; r_2))$, where $\forall i \in [n]$: $\tilde{c}_i = \left(c_i^1, c_i^1, \ldots, c_i^{\mathsf{d}_{\mathsf{Pack}}}\right)$. We recall that in the encryption, we divide the shares $[\![s]\!]_i \in \{0,1\}^m$ into $\lceil \frac{m}{\mathsf{d}_{\mathsf{Pack}}} \rceil$ parts $[\![s]\!]_{i,1}, \ldots, [\![s]\!]_{i,\mathsf{d}_{\mathsf{Pack}}} \in \{0,1\}^{\lceil \frac{m}{\mathsf{d}_{\mathsf{Pack}}} \rceil}$ and randomly encode each of them $[\![\hat{s}]\!]_{i,j} \leftarrow \mathsf{REnc}([\![s]\!]_{i,j}, a_j, N) \in \mathbb{Z}_N^*$ as a random Paillier plaintext and we use to follow the equality test predicate to compute $(c_i^j)_{\forall 1 \leq j \leq \mathsf{d}_{\mathsf{Pack}}}$. Here, for all $1 \leq j \leq \mathsf{d}_{\mathsf{Pack}} - 1$ we have $a_j = 2^{\lceil \frac{m}{\mathsf{d}_{\mathsf{Pack}}} \rceil}$ and $a_{\mathsf{d}_{\mathsf{Pack}}} = 2^{m - \mathsf{d}_{\mathsf{Pack}} \lceil \frac{m}{\mathsf{d}_{\mathsf{Pack}}} \rceil + \lceil \frac{m}{\mathsf{d}_{\mathsf{Pack}}} \rceil}$. So, in the conditional decryption algorithm, after decryption of all the Paillier ciphertexts associated to the same share, i.e., $\lceil \hat{s}' \rceil_{i,j} = \mathsf{P.Dec}_{sk}(c_i^j)$, we decode them to $[\![s']\!]_{i,j} = [\![\hat{s}']\!]_{i,j} \bmod a_j$ and combine all $\mathsf{d}_{\mathsf{Pack}}$ together to reconstruct its corresponding share $[\![s']\!]_i \in \{0,1\}^m$. Since the (de)packing scheme is bijective, for all uncorrupted indices we will recover valid shares, and for all corrupted indices we obtain a randomly corrupted share. Now, without loss of generality, we continue with the recovered shares.

Now, let $S^* = \{i \in [n] \ : \ m_2[i] = m_1[i]\}$ denote the indices of the characters where m_2 and m_1 match. For analytical simplicity, we initially assume that for all $i \in [n] \setminus S^*$, the shares $[\![s]\!]_i \in_R \{0,1\}^m$ are drawn uniformly at random. We subsequently extend the proof to the actual distribution. So, if $P_{\mathtt{Ham},\ell}(m_1, m_2) = 1$, we have $|S^*| \geq n - \ell$ and, due to $(n, t = n - \ell, 2^{-n\lambda_1})$-robustness of our RRSS scheme, we have

$$\begin{aligned} &\Pr\left[S \subseteq S^* \wedge |S| \geq n - \ell \,\middle|\, S = \Pi_{\mathsf{RRSS}}.\mathsf{Identify}\left(\left([\![s']\!]_i\right)_{\forall i \in [n]}, \mathsf{pp}\right)\right] \\ &\quad \geq 1 - 2^{-\lambda} \end{aligned}$$

where the probability is taken over the uniformly random corruptions of the shares $[\![s']\!]_i$ for $i \notin S^*$. As long as $S \subseteq S^*$ we are guaranteed to recover the correct symmetric key i.e.,

$$K = K_S = \Pi_{\mathsf{RRSS}}.\mathsf{SecretRecover}\left(\left\{(n, k = n - \ell, [\![s']\!]_i)_{i \in S}\right\}, \mathsf{pp}\right) \ .$$

From the correctness of the authenticated encryption scheme it follows that $\mathsf{Auth.Dec}_{K_S}(c_{AE}) = m_3$.

In the previous paragraph, we assumed that the value $x_i = [\![s']\!]_i \in_R \{0,1\}^m$ is uniformly random for each $i \notin S^*$. This is close, but it is not quite true. In reality, the distribution of $x_i = [\![s']\!]_i$ is only close to the uniform distribution. According to the conditional encryption algorithm, we observe that for $i \notin S^*$ the distribution of $[\![s']\!]_i \in \{0,1\}^m$ is as follows, sample uniformly random vector $(\hat{y}_0, \hat{y}_1, \ldots, \hat{y}_{\mathsf{d}_{\mathsf{Pack}}}) \in_R \mathbb{Z}_N^{\mathsf{d}_{\mathsf{Pack}}}$ and $\forall j \leq \mathsf{d}_{\mathsf{Pack}}$ compute $\hat{y}'_j = \hat{y}_j \bmod a_j$ (and represent it as a $\log_2 a_j$-bit number and when we concatenate all these bits strings together we get $\hat{y}'_1 \| \ldots \| \hat{y}'_{\mathsf{d}_{\mathsf{Pack}}} \in \{0,1\}^m$ which is m-bit string. By applying the results of Theorem 15 with $a_0 = a_j$ and $k_0 = \lfloor \frac{N}{a_j} \rfloor - 1$, the statistical distance between original/modified distributions of our recovered shares $x_1, \ldots, x_n$ is upper bounded by $|S| \left(\mathsf{d}_{\mathsf{Pack}} \frac{1}{k_0 + 1}\right) \leq \frac{n \mathsf{d}_{\mathsf{Pack}} a_0}{N} \leq \frac{n \mathsf{d}_{\mathsf{Pack}} 2^{\lceil \frac{m}{\mathsf{d}_{\mathsf{Pack}}} \rceil}}{N}$. By choice of $\mathsf{d}_{\mathsf{Pack}}$ we have $\frac{n \mathsf{d}_{\mathsf{Pack}} 2^{\lceil \frac{m}{\mathsf{d}_{\mathsf{Pack}}} \rceil}}{N} \leq 2^{-\lambda}$. It follows that

$$\begin{aligned} &\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c_1, m_2, m_3; r_2\right)\right) = m_3 \middle| c_1 = \mathsf{Enc}_{pk}\left(m_1; r_1\right)\right] \\ &\quad \geq 1 - 2^{-\lambda} - 2^{-\lambda} = 1 - \epsilon_1(\lambda) \ . \end{aligned}$$

Error Detection for Conditional Ciphertexts: We will show that for all messages $m_1, m_2 \in \Sigma^n$ such that $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 0$, all payload messages m_3, all $\{sk, pk\}$ in the support of our Key Generation algorithm

and all random strings $r_1 \in_R (\mathbb{Z}_N^*)^n$ we have

$$\begin{aligned}&\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c_1, m_2, m_3; r_2\right)\right) = \perp | c_1 = \mathsf{Enc}_{pk}\left(m_1; r_1\right)\right] \\ &\quad \geq 1 - 2^{-n\lambda} - 2^{-\lambda} \geq 1 - \epsilon(\lambda)\end{aligned}$$

where the randomness is taken over the random coins r_2 of the conditional encryption algorithm CEnc.

As above we let $S^* \subseteq [n]$ denote the correct indices where m_1 and m_2 match. Because the predicate $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 0$ we have $|S^*| < n - \ell$. For simplicity we will first assume that for each $i \notin S^*$ the corresponding share $[\![s']\!]_i$ is uniformly randomly corrupted. By perfect privacy the uncorrupted shares in S^* are also distributed uniformly at random. For each fixed set $S \subseteq [n]$ of size $|S| = n - \ell$ we can view

$$K_S = \Pi_{\mathsf{RRSS}}.\mathsf{SecretRecover}\left(\left\{(n, k = n - \ell, [\![s']\!]_i)_{i \in S}\right\}, \mathsf{pp}\right) \ ,$$

as uniformly random. If K_S is uniformly random then $\Pr[\mathsf{Auth.Dec}_{K_S}(c_{AE}) \neq \perp] \leq \epsilon_{AE}(\lambda)$. Union bounding over all possible $S \subseteq [n]$ of size $|S| = n - \ell$ we have

$$\Pr\left[\exists S \subseteq [n].|S| = n - \ell \wedge \mathsf{Auth.Dec}_{K_S}(c_{AE}) \neq \perp\right] \leq \binom{n}{\ell} \epsilon_{AE}(\lambda) \ .$$

If K_S fails to decrypt c_{AE} for all possible S then the conditional decryption algorithm must output $\perp$ as $\Pi_{\mathsf{RRSS}}.\mathsf{Identify}\left(([\![s']\!]_i,)_{\forall i \in [n]}, \mathsf{pp}\right)$ will either output $\perp$ or a set S of size $|S| = n - \ell$.

In the previous paragraph, we assumed that the value $x_i = [\![s']\!]_i \in_R \{0,1\}^m$ is uniformly random for each $i \notin S^*$. This is close, but it is not quite true. In reality, the distribution of $x_i = [\![s']\!]_i$ is close to the uniform distribution. According to the conditional encryption algorithm, we observe that for $i \notin S^*$ the distribution of $[\![s']\!]_i \in \{0,1\}^m$ is as follows, sample uniformly random vector $(\hat{y}_0, \hat{y}_1, \ldots, \hat{y}_{\mathsf{d_{Pack}}}) \in_R \mathbb{Z}_N^{\mathsf{d_{Pack}}}$ and $\forall j \leq \mathsf{d_{Pack}}$ compute $\hat{y}'_j = \hat{y}_j \bmod a_j$ (and represent it as a $\log_2 a_j$-bit number and when we concatenate all these bits strings together we get $\hat{y}'_1 \| \ldots \| \hat{y}'_{\mathsf{d_{Pack}}} \in \{0,1\}^m$ which is m-bit string. By applying the results of Theorem 15 with $a_0 = a_j$ and $k_0 = \lfloor \frac{N}{a_j} \rfloor - 1$, the statistical distance between original/modified distributions of our recovered shares $x_1, \ldots, x_n$ is upper bounded by $|S| \left(\mathsf{d_{Pack}} \frac{1}{k_0 + 1}\right) \leq \frac{n \mathsf{d_{Pack}} a_0}{N} \leq \frac{n \mathsf{d_{Pack}} 2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}}{N} \leq 2^{-\lambda}$. It follows that

$$\begin{aligned}&\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c_1, m_2, m_3; r_2\right)\right) = \perp | c_1 = \mathsf{Enc}_{pk}(m_1; r_1)\right] \\ &\quad \geq 1 - 2^{-\lambda} - \binom{n}{\ell} \epsilon_{AE}(\lambda) \geq 1 - \epsilon(\lambda) \ .\end{aligned}$$

Error Detection for Conditional Ciphertexts with Dealer Private RRSS Scheme: Assuming that the RRSS sharing scheme is $(n - \ell - 1, \epsilon_2(\lambda))$-dealer private we will show that for all messages $m_1, m_2 \in \Sigma^n$ such that $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 0$, all payload messages m_3, all $\{sk, pk\}$ in the support of our Key Generation algorithm and all random strings $r_1 \in_R (\mathbb{Z}_N^*)^n$ we have

$$\begin{aligned}&\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c_1, m_2, m_3; r_2\right)\right) = \perp | c_1 = \mathsf{Enc}_{pk}\left(m_1; r_1\right)\right] \\ &\quad \geq 1 - \epsilon_2(\lambda) - 2^{-\lambda} \geq 1 - \epsilon(\lambda) \ .\end{aligned}$$

where the randomness is taken over the random coins r_2 of the conditional encryption algorithm CEnc.

Let $S^* = \{i : m_1[i] = m_2[i]\}$ and observe that $|S^*| \leq n - \ell - 1$. The remaining shares $[\![s']\!]_i$ for $i \notin S^*$ are statistically close to the uniform distribution. For now let us view $[\![s']\!]_i$ as uniformly random for each $i \notin S^*$. Then by $(n - \ell - 1, \epsilon_2(\lambda))$-dealer privacy we have

$$\Pr\left[\Pi_{\mathsf{RRSS}}.\mathsf{Identify}\left(([\![s']\!]_i)_{\forall i \in [n]}, \mathsf{pp}\right) = \perp\right] \geq 1 - \epsilon_2(\lambda) \ .$$

As long as $\Pi_{\mathsf{RRSS}}.\mathsf{Identify}\left(\left(\llbracket s' \rrbracket_i\right)_{\forall i\in[n]}, \mathsf{pp}\right) = \perp$ the conditional decryption algorithm will certainly output $\perp$.

In the previous paragraph, we assumed that the value $x_i = \llbracket s' \rrbracket_i \in_R \{0,1\}^m$ is uniformly random for each $i \notin S^*$. This is close, but it is not quite true. In reality, the distribution of $x_i = \llbracket s' \rrbracket_i$ is close to the uniform distribution. According to the conditional encryption algorithm, we observe that for $i \notin S^*$ the distribution of $\llbracket s' \rrbracket_i \in \{0,1\}^m$ is as follows, sample uniformly random vector $(\hat{y}_0, \hat{y}_1, \ldots, \hat{y}_{\mathsf{d_{Pack}}}) \in_R \mathbb{Z}_N^{\mathsf{d_{Pack}}}$ and $\forall j \leq \mathsf{d_{Pack}}$ compute $\hat{y}'_j = \hat{y}_j \bmod a_j$ (and represent it as a $\log_2 a_j$-bit number and when we concatenate all these bits strings together we get $\hat{y}'_1 \| \ldots \| \hat{y}'_{\mathsf{d_{Pack}}} \in \{0,1\}^m$ which is m-bit string. By applying the results of Theorem 15 with $a_0 = a_j$ and $k_0 = \lfloor \frac{N}{a_j} \rfloor - 1$, the statistical distance between original/modified distributions of our recovered shares $x_1, \ldots, x_n$ is upper bounded by $|S| \left(\mathsf{d_{Pack}} \frac{1}{k_0+1}\right) \leq \frac{n\mathsf{d_{Pack}} a_0}{N} \leq \frac{n\mathsf{d_{Pack}} 2^{\lceil \frac{m}{\mathsf{d_{Pack}}} \rceil}}{N} \leq 2^{-\lambda}$. It follows that

$$\begin{aligned}&\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c_1, m_2, m_3; r_2\right)\right) = \perp | c_1 = \mathsf{Enc}_{pk}\left(m_1; r_1\right)\right] \\ &\quad \geq 1 - 2^{-\lambda} - \epsilon_2(\lambda)\end{aligned}$$

□

H.5 Proof of ROR security of Construction 2

We also prove that **Construction 2** provides ROR security.

Theorem 17 (ROR security of Construction 2) *Assume that the Paillier encryption scheme is $(t_{\mathsf{P}}(\lambda), q_{\mathsf{P}}(\lambda), \epsilon_{\mathsf{P}}(t_{\mathsf{P}}(\lambda), q_{\mathsf{P}}(\lambda), \lambda))$-Real-Or-Random secure and that Π_{AE} is $(t_{AE}(\lambda), q_{AE}(\lambda), \epsilon_{AE}(t_{AE}(\lambda), q_{AE}(\lambda), \lambda))$-Real-Or-Random. Then **Construction 2** is $(t'(\lambda), q'(\lambda), \epsilon'(t'(\lambda), q'(\lambda), \lambda))$-ROR[8] secure with $t'(\lambda) = \min\{t_{\mathsf{P}}(\lambda), t_{AE}(\lambda)\} - O(n)$ and $\epsilon'(t'(\lambda), q'(\lambda), \lambda) = 2n(\mathsf{d_{Pack}}+1)\epsilon_{\mathsf{P}}(t_{\mathsf{P}}(\lambda), q_{\mathsf{P}}(\lambda), \lambda) + 2\epsilon_{AE}(t_{AE}(\lambda), q_{AE}(\lambda), \lambda)$ and $q'(\lambda) = \min\{\frac{q_{\mathsf{P}}(\lambda)}{2n(\mathsf{d_{Pack}}+1)}, \frac{q_{AE}(\lambda)}{2}\}$.*

Proof of Theorem 17: (Sketch) Let $c_1 \in \mathbb{Z}_{N^2}$ (we do not assume $c_1 \in \mathbb{Z}^*_{N^2}$). And consider the query (c_1, m_2, m_3) to the encryption oracle. If $b = 1$, then we have $(b = 1, \mathsf{d_{Pack}}, c, c_{AE})$ in which $c = (c[1], \ldots, c[n]) \leftarrow \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ where $c[i] = (c[i,1], \ldots, c[i, \mathsf{d_{Pack}}])_{\forall i\in[n]}$ and for all $j \in [\mathsf{d_{Pack}}]$ we have $c[i,j] = c_1[i]^{R_{i,j}}((N+1)^{-m_2[i]} r_{1,i,j}^N)^{R_{i,j}}(1+N)^{p_{i,j}} r_{2,i,j}^N$ for uniformly random $R_{i,j} \in \mathbb{Z}_N$ and $r_{1,i,j}, r_{2,i,j} \in_R \mathbb{Z}_N^*$. By ROR security of our Paillier encryption scheme, we can apply a Hybrid argument, and can swap $(1+N)^{-m_2[i]} r_{1,i,j}^N$ (resp. $(1+N)^{p_{i,j}} r_{2,i,j}^N$) with $(1+N)^{-r_1[i]} r_{1,i,j}^N$ (resp. $(1+N)^{-r_2[i]} r_{2,i,j}^N$) with $r_1[i], r_2[i] \in_R \mathbb{Z}_N$. After $n + n\mathsf{d_{Pack}} = n(\mathsf{d_{Pack}}+1)$ hybrids, we have replaced each $c[i,j]$ with a ciphertext of the form $c_1[i]^{R_{i,j}}(1+N)^{-r_1[i]}(r_{1,i,j})^N)^{R_{i,j}}(1+N)^{r_2[i]} r_{2,i,j}^N \bmod N^2$. In particular, all information about the secret symmetric key K is removed so we can invoke ROR security for Authenticated encryption to replace c_{AE} with a random ciphertext. We can then reverse the hybrids to move to transition from $\mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ (when $b = 0$) all the way to $b = 1$ case $\mathsf{CEnc}_{pk}(c_1, m'_2, m'_3)$ for random messages m'_2, m'_3. Thus, we adjust $q'(\lambda) = q_{\mathsf{P}}(\lambda)/(2n(\mathsf{d_{Pack}}+1))$ since we invoke ROR security for Paillier during $(2n(\mathsf{d_{Pack}}+1)) \times q'(\lambda)$ hybrids. Similarly, we invoke ROR security for authenticated encryption in $q_{AE}(\lambda) = 2q'(\lambda)$ hybrids. So, we can set $q'(\lambda) = \min\{\frac{q_{\mathsf{P}}(\lambda)}{2n(\mathsf{d_{Pack}}+1)}, \frac{q_{AE}(\lambda)}{2}\}$. Applying the union bound, we have $\epsilon'(t'(\lambda), q'(\lambda), \lambda) = 2n(\mathsf{d_{Pack}}+1)\epsilon_{\mathsf{P}}(t_{\mathsf{P}}(\lambda), q_{\mathsf{P}}(\lambda), \lambda) + 2\epsilon_{AE}(t_{AE}(\lambda), q_{AE}(\lambda), \lambda)$ where $t'(\lambda) = \min\{t_{\mathsf{P}}(\lambda), t_{AE}(\lambda)\} - O(n)$. □

H.5.1 Useful Lemma for Equivalency of the share Distributions after Packing the RRSS Shares

After packing and unpacikng the output of our RRSS scheme, one may be wondering if the resulting distribution is the same os the shares distribution or not. Since we using one-to-one mapping, the distribution will remain the same even after packing/unpacking. So for the sake of completeness we provide the formal lemma and its proof to support our argument.

[8] We use ROR instead of Real-Or-Randonm for simplicity.

Lemma 3 (Substrings of a Uniform Binary String are Uniform) *Let $x = ab + r \geq 1$ s.t. $r < 0 \leq b$, $\mathsf{MaxM} = 2^x = 2^{ad+r}$ be integers. Let $R_{bin} \in_R \{0,1\}^{ad+r}$ be x-bit binary string chosen uniformly at random and its corresponding decimal integer be the uniformly random number $R \in \mathbb{Z}_{2^x}$. We write R as follows using a vector $a+1$ elements $(d_0, \ldots, d_{a-1}, d_a)$ such that $(d_i)_{\forall 1 \leq i \leq a-1} \in \mathbb{Z}_{2^d}$ and $d_a \in \mathbb{Z}_{2^r}$ as follows:*

$$R \;=\; \sum_{i=0}^{a-2} d_i 2^i b + r_a 2^{(a-1)d+r}, \qquad \forall 0 \leq i \leq a-1 : d_i \in \mathbb{Z}_{2^d}, d_a \in \mathbb{Z}_{2^r}.$$

Then the digits $(d_0, \ldots, d_a)$ are independent and, for each $0 \leq i \leq a$, d_i is uniform on $\mathbb{Z}_{2^d}$ and $d_a \in_R \mathbb{Z}_{2^r}$.

Proof of Lemma 3: Consider the map

$$f: \; \{0,1,\ldots,2^d-1\}^a \times \{0,1,\ldots,2^r-1\} \longrightarrow \{0,1,\ldots,2^x-1 = 2^{ad+r}-1\},$$
$$f(d_0,\ldots,d_a) \;=\; \left(\sum_{i=0}^{a-2} d_i 2^{di}\right) + d_a 2^{d(a-1)+r}.$$

We note that as all 2^x possible element are mapped one-to-one, then f is a bijection.

Since R is uniform on $\{0,\ldots,2^x-1\}$ and $(d_0,\ldots,d_a) = f^{-1}(R)$, it follows that $(d_0,\ldots,d_a)$ is uniform on the 2^{ad+r}-element set $\{0,\ldots,2^d-1\}^a \times \{0,\ldots,2^r-1\}$.

Fix any index $0 \leq i \leq a-1$ and any $v \in \{0,\ldots,2^d-1\}$. The number of $a+1$-tuples with i-th coordinate equal to v is $2^{d(a-1)} \times 2^r$, hence

$$\Pr[d_i = v] = \frac{|\{(d_0,\ldots,d_a) : d_i = v\}|}{2^{ad+r}} = \frac{2^{d(a-1)} \times 2^r}{2^{ad+r}} = \frac{1}{2^d},$$

so each d_i is uniform on $\{0,\ldots,2^d-1\}$.

With similar argument we can show that the $a+1$-th index $d_a \in \{0, 2^r-1\}$ is also chosen uniformly at random and we have

$$\Pr[d_a = v] = \frac{|\{(d_0,\ldots,d_a) : d_a = v\}|}{2^{ad+r}} = \frac{2^{d(a)}}{2^{ad+r}} = \frac{1}{2^r}\ .$$

which shows the digits are independently chosen uniformly at random. □

I Constructions from Prior Work for Comparison

We highlight that fPAKE protocol from [BFH+23] (Figure 11) provides the relaxed security definition. The relaxed security definition is identical to the ideal functionality in Figure 3 except that the test password oracle TestPwd is replaced with the following test password query which uses three following leakage functions L_c, L_m, L_f for two n characters passwords $\mathtt{pwd} = (\mathtt{pwd}_1, \ldots, \mathtt{pwd}_n)$ and $\mathtt{pwd}' = (\mathtt{pwd}'_1, \ldots, \mathtt{pwd}'_n)$:

- $L_c(\mathtt{pwd}, \mathtt{pwd}') := \left(\left\{j : \text{ s.t. } \mathtt{pwd}_j = \mathtt{pwd}'_j\right\}, \text{“correct guess”}\right)$.
- $L_m(\mathtt{pwd}, \mathtt{pwd}') := \left(\left\{j \text{ s.t. } \mathtt{pwd}_j = \mathtt{pwd}'_j\right\}, \text{“wrong guess”}\right)$
- $L_f(\mathtt{pwd}, \mathtt{pwd}') :=$ “wrong guess.

Now we describe the weaker notion of test password as follows:

- **TestPwd**: On $(\text{TestPwd}, \mathsf{sid}, \mathsf{Party}_i, \mathtt{pwd}^*)$ from the adversary $\mathcal{A}$: If there is a `fresh` record $(\mathsf{Party}_i, \mathtt{pwd}^{\mathsf{Party}_i})$, then set $\ell' \leftarrow \mathtt{Ham}(\mathtt{pwd}_{\mathsf{Party}_i}, \mathtt{pwd}^*)$ and do:

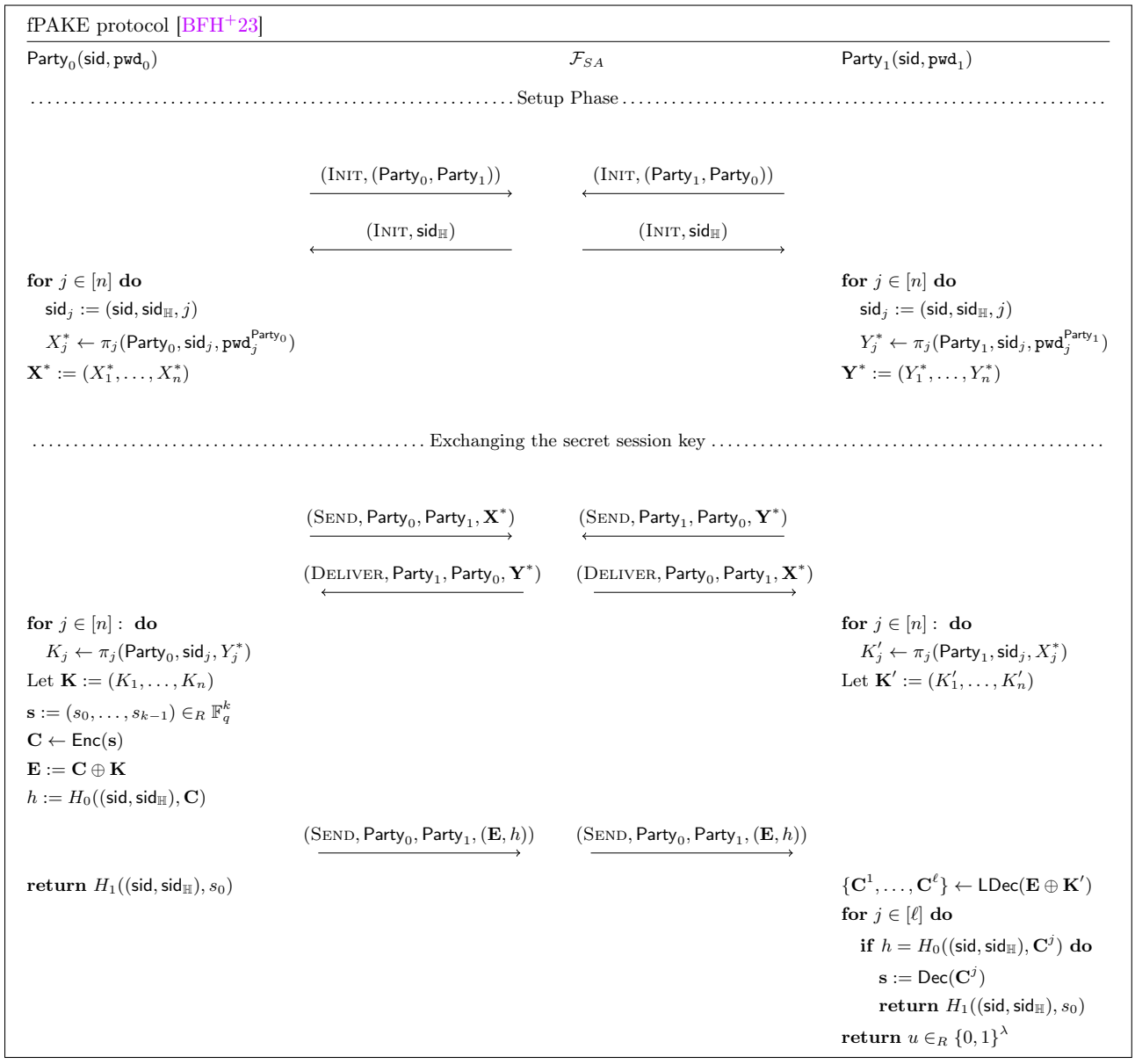


Figure 11: Description of π_{fPAKE} protocol from [BFH+23] that uses n executions of the protocol π_j (realizing $\mathcal{F}_{\mathsf{iPAKE}}$ in Figure 6) in the $\mathcal{F}_{\mathsf{SA}}$-hybrid model and random encoding scheme with list decoding $\Pi_{\textsc{RandEncode}} = (\mathsf{Enc}, \mathsf{Dec}, \mathsf{LDec})$. The protocol π_{fPAKE} only achieves a relaxed notion of fPAKE security.

Construction 3

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda; r)$:

1. run $(\mathtt{P}.sk, \mathtt{P}.pk = (N)) \leftarrow \mathtt{P}.\mathsf{KeyGen}(1^\lambda; r)$.
   - If $\min\{p, q\} < |\Sigma|$ or $\gcd(N, \phi(N)) \neq 1$ repeat step 1. //run P.KeyGen again.
3. set $\mathtt{SS} = \Pi_{SSS}.(\mathsf{ShareGen}, \mathsf{SecretRecover})$ over finite field $\mathbb{F}_{2^\lambda}$.
2. set $sk = \mathtt{P}.sk$ and $pk = (\mathtt{P}.pk, \lambda)$.

$c \leftarrow \mathsf{Enc}_{pk}(m; r)$:

1. parse $m = (m[1], \ldots, m[n]), r = (r_1, \ldots, r_n) \in (\mathbb{Z}_N^*)^n$.
2. $\forall 1 \leq i \leq n$:
   - compute $\hat{m}[i] = \mathsf{ToInt}(m[i])$
   - compute $c_i = (1+N)^{\hat{m}[i]} {r_i}^N \mod N^2$
3. output $c = (b = 0, c_1, \ldots, c_n)$.

$c \leftarrow \mathsf{CEnc}_{pk}(c_m, m_2, m_3; r)$:

1. parse $m_2 = (m_2[1], \ldots, m_2[n]) \in \Sigma^n$, $r = (r_1, \ldots, r_n) \in (\mathbb{Z}_N^*)^n$, $c_m = (b, c_1, \cdots, c_n)$, $pk = (\mathtt{P}.pk, \lambda)$;
2. if $b = 1$, output $\perp$.
4. $\forall i \in [n]$: compute $\hat{m}_2[i] = \mathsf{ToInt}(m_2[i])$;
5. Pick $K \in_R \{0,1\}^\lambda$, compute $c_{AE} = \mathsf{Auth.Enc}_K(m_3)$;
6. $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \mathtt{SS}.\mathsf{ShareGen}_\lambda(n, n-l, K)$;
7. $\forall j \in [n]$, compute $\llbracket \hat{s} \rrbracket_j = \llbracket s \rrbracket_j + 2^\lambda k_j$ s.t. $k_j \in_R \left[\lfloor \frac{N}{2^\lambda} \rfloor \right]$;
8. $\forall j \in [n]$, compute $\tilde{c}[j] = c_j^{-R_j} (N+1)^{R_j m_2[j] + \llbracket \hat{s} \rrbracket_j}$
9. output $\tilde{c} = (b = 1, \tilde{c}[1], \cdots, \tilde{c}[n], c_{AE})$;

$\{m, \perp\} = \mathsf{CDec}_{sk}((b, \tilde{c}))$:

2.1 if $b = 1$ // The ciphertext is extracted from CEnc(·)
   (1) parse $\tilde{c} = (\tilde{c}[1], \cdots, \tilde{c}[n], c_{AE})$, AND set $m = \perp$
   (2) $\forall j \in [n]$compute $\llbracket \hat{s}' \rrbracket_j = \mathtt{P}.\mathsf{Dec}_{sk}(c[j])$ and compute $\llbracket s' \rrbracket_j = \llbracket \hat{s}' \rrbracket_j \bmod 2^\lambda$.
   (3) For all $\binom{n}{n-\ell}$ possible $S \subseteq [n]$ with $|S| = n - \ell$
      - compute $K_S = \mathtt{SS}.\mathsf{SecretRecover}\left(\{\llbracket s' \rrbracket_j\}_{\forall j \in S}\right)$
      - compute $(v, \hat{m}') = \mathsf{Auth.Dec}_{K_S}(c_{AE})$, if $v = 1$, return $m = \hat{m}'$
   (4) If for all $S \subseteq [n]$, $v = 0$, then return $m = \perp$.

2.2 if $b = 0$ // The ciphertext is extracted from Enc(·)
   - parse $\tilde{c} = (\tilde{c}_1, \cdots, \tilde{c}_n)$, AND set $m = \perp$
   - for $1 \leq i \leq n$, compute $\hat{m}_i = \mathtt{P}.\mathsf{Dec}_{sk}(\tilde{c}_i)$
   - return $m = \mathsf{ToInt}^{-1}(\hat{m}_1) \| \ldots \| \mathsf{ToInt}^{-1}(\hat{m}_n)$

Figure 12: Ameri and Blocki's [AB24] concrete construction of conditional encryption over Hamming distance at most ℓ, i.e., $P_{\ell,\mathtt{Ham}}$. Note that the running time of step 3 in CDec is $\Omega\left(\binom{n}{\ell}\right)$ which intractable for non-constant $\ell = \omega(1)$.

- If $\ell' \leq \ell$, mark the record `compromised` and reply to $\mathcal{A}$ with the leakage $L_c(\mathtt{pwd}^{\mathsf{Party}_i, \mathtt{pwd}^*})$;
- If $\ell < \ell' \leq \gamma$, mark the record `compromised` and reply $L_m(\mathtt{pwd}^{\mathsf{Party}_i}, \mathtt{pwd}^*)$.
- $\gamma < \ell'$, mark the record `interrupted` and replay $\mathcal{A}$ with $L_f(\mathtt{pwd}^{\mathsf{Party}_i}, \mathtt{pwd}^*)$.

Here ℓ denotes the Hamming distance threshold, and $n - \gamma$ represents the robustness parameter. In prior work on robust secret sharing [DHP+18, BFH+23] with perfect privacy, there exists a gap between the reconstruction threshold $t = n - \ell$ and the robustness bound $n - \gamma$. This gap corresponds to a strictly weaker security notion: when the Hamming distance between two passwords lies in the interval $(\ell, \gamma]$, the ideal functionality may still reveal the positions of matching characters, even though the key-exchange will be unsuccessful. In contrast, our RRSS construction eliminates this gap by ensuring that the reconstruction and robustness thresholds coincide. Consequently, we adopt the original TestPwd oracle, which does not include any auxiliary leakage functionality.

I.1 Additional Comments and Discussion

I.2 Impossibility of Strong Robustness and Perfect Privacy with No Gap

As we noted in the introduction it is impossible to have a (t, n) secret sharing scheme that that preserves $(t-1)$-perfect privacy and is strongly robust up to the corrupting of $n - t$ shares.

More precisely, we say that a (t, n) secret sharing scheme is strongly robust if for all secrets s, all

vectors $[\![s]\!]_1, \ldots, [\![s]\!]_n$ in the support of $\mathsf{ShareGen}(s)$ and all nearby vectors $[\![s']\!]_1, \ldots, [\![s']\!]_n$ such that $|\{i : [\![s']\!]_i = [\![s]\!]_n\}| \geq t$ we have $\mathsf{SecretRecover}([\![s']\!]_1, \ldots, [\![s']\!]_n) = s$.

Claim 18 *If a (t, n) secret sharing scheme satisfies $(t-1)$-perfect privacy then it is not strongly robust.*

Proof : Suppose that our secret sharing scheme is $(t-1)$-perfectly private and let $[\![s]\!]_1, \ldots, [\![s]\!]_n$ be shares of a secret s i.e., in the support of $\mathsf{ShareGen}(s)$. Now consider generating shares for a different secret s'. Because the first $(t-1)$-shares are distributed uniformly we can find shares $[\![s']\!]_1, \ldots, [\![s']\!]_n$ of s' (i.e., in the support of $\mathsf{ShareGen}(s')$) which match our shares for s on the first $t-1$ indices i.e., $[\![s']\!]_i = [\![s]\!]_i$ for all $i \leq t-1$. Now define $[\![s'']\!]_i = [\![s]\!]_i$ for all $i \leq (t-1)+(n-t+1)/2$ and $[\![s'']\!]_i = [\![s']\!]_i$ for all $i \geq t+(n-t+1)/2$. Observe that $[\![s'']\!]_1, \ldots, [\![s'']\!]_n$ matches $[\![s]\!]_1, \ldots, [\![s]\!]_n$ on at least $(t-1)+(n-t+1)/2 \geq t$ indices. Thus, by strong robustness we have $\mathsf{SecretRecover}([\![s'']\!]_1, \ldots, [\![s'']\!]_n) = s$. However, we also observe that $[\![s'']\!]_1, \ldots, [\![s'']\!]_n$ matches $[\![s]\!]_1, \ldots, [\![s]\!]_n$ on at least $(t-1)+(n-(t-1)-(n-(t-1))/2) \geq t$ indices. Thus, by strong robustness we should also have $\mathsf{SecretRecover}([\![s'']\!]_1, \ldots, [\![s'']\!]_n) = s''$. Contradiction! □

Remark 19 *We could also consider a weaker notions of robustness where the adversary is allowed to corrupt up to $n-t$ shares, but the adversary does not get to view* any *of the other $n-t$ shares. This is commonly referred to as the "non-rushing" model. Even under this weaker notion of robustness it is impossible to achieve robustness when $t \leq n/2$ as this would mean that there can be more corrupted shares than honest shares i.e., $n-t \geq n/2 \geq t$ [IOS12]. For example, if $n-t \geq t$ then the adversary could simply replace these $n-t$ shares with honestly generated shares for an entirely different secret s'. Now it is unclear whether reconstruction should output our original secret s or the new secret s'. On the one hand at least t shares correspond to our original secret s, but at least $n-t \geq t$ shares correspond to a different secret s'. By contrast, it is possible to achieve random robustness when $t \leq n/2$.*

I.3 Discussion about our fPAKE Protocol

Remark 20 *It is natural to wonder why we use $\mathcal{H}((\mathsf{sid}, \mathsf{sid}_{\mathbb{H}}), s)$ instead of s when s is already uniformly random. The reason is that this prevents the attack of [BFH+23] on the fPAKE protocol of [DHP+18]. Here an attacker plays man-in-the-middle attacker but only for one of the* iPAKE *executions. Because altering a single character of the password may alter the final outcome of the* fPAKE *and this final outcome (success or failure) is visible to the attacker, the attacker can exploit this vulnerability to guess one character of the password at a time leading to a much more efficient password recovery attack. Following [BFH+23] we use the split authentication functionality [BCL+05] $\mathcal{F}_{\mathsf{SA}}$ along with the random oracle $\mathcal{H}$ to prevent this attack. If the attacker plays man-in-the-middle in* any iPAKE *execution then both honest parties will have different $\mathsf{sid}_{\mathbb{H}}$ resulting in different keys.*