# Conditional Encryption with Applications to Secure Personalized Password Typo Correction

Mohammad Hassan Ameri Jeremiah Blocki

Purdue University
West Lafayette, IN USA

**Abstract**

We introduce the notion of a conditional encryption scheme as an extension of public key encryption. In addition to the standard public key algorithms ($\mathsf{KeyGen}$, $\mathsf{Enc}$, $\mathsf{Dec}$) for key generation, encryption and decryption, a conditional encryption scheme for a binary predicate $P$ adds a new conditional encryption algorithm $\mathsf{CEnc}$. The conditional encryption algorithm $c = \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ takes as input the public encryption key $pk$, a ciphertext $c_1 = \mathsf{Enc}_{pk}(m_1)$ for an unknown message $m_1$, a control message $m_2$ and a payload message $m_3$ and outputs a conditional ciphertext $c$. Intuitively, if $P(m_1, m_2) = 1$ then the conditional ciphertext $c$ should decrypt to the payload message $m_3$. On the other hand if $P(m_1, m_2) = 0$ then the ciphertext should not leak *any* information about the control message $m_2$ or the payload message $m_3$ even if the attacker already has the secret decryption key $sk$. We formalize the notion of conditional encryption secrecy and provide concretely efficient constructions for a set of predicates relevant to password typo correction. Our practical constructions utilize the Paillier partially homomorphic encryption scheme as well as Shamir Secret Sharing. We prove that our constructions are secure and demonstrate how to use conditional encryption to improve the security of personalized password typo correction systems such as TypTop. We implement a C++ library for our practically efficient conditional encryption schemes and evaluate the performance empirically. We also update the implementation of TypTop to utilize conditional encryption for enhanced security guarantees and evaluate the performance of the updated implementation.

**Full Version:** This document provides the full version of the article published at CCS 2024 under the same title – see https://doi.org/10.1145/3658644.3690374.

## 1 Introduction

Traditionally, public key encryption allows any party who has the public key $pk$ to encrypt a message $m$ and obtain a ciphertext $c$ which can only be decrypted by a party who possesses the corresponding secret key $sk$. The implicit assumption is that anyone who posses the secret key $sk$ is a trusted party. However, there are some applications where the party encrypting a message may only want to conditionally reveal that message to the party who possesses the secret key if certain conditions hold. For example, consider the problem of having an authentication server maintain an (encrypted) cache of incorrect login attempts for each user. Such a cache might be used to design a (personalized) typo tolerant password authentication scheme [CAA+16, CWP+17] and/or to help identify malicious login attempts. In the TypTop [CWP+17] system the sever generates a public/private key pair $(sk_u, pk_u)$ for each user account $u$ and encrypts the secret key $sk_u$ with a symmetric encryption key $K_u = \mathtt{PKDF}(s_u, \mathtt{pwd}_u)$ derived from the user's password $\mathtt{pwd}_u$ and a random salt value $s_u$. The server then stores the resulting ciphertext $c_u = \mathsf{Enc}_{K_u}(sk_u)$ and public key $pk_u$ along with the salt value $s_u$. Whenever the user logs in with an incorrect password $\mathtt{pwd}'$ the authentication server uses the public key $pk_u$ to generate and store the ciphertext $c' = \mathsf{Enc}_{pk_u}(\mathtt{pwd}')$. Later if the user logs in with a correct password $\mathtt{pwd}$ we can re-derive the symmetric key $K_u = \mathtt{PKDF}(s_u, \mathtt{pwd})$, use the symmetric key to recover the secret key $sk_u = \mathsf{Dec}_{K_u}(c_u)$ and then use the secret key $sk_u$ to decrypt each

password $\texttt{pwd}' = \mathsf{Dec}_{sk_u}(c')$ in our encrypted cache. The TypTop system can then examine each particular password $\texttt{pwd}'$ in the cache to determine whether or not this password is an "acceptable typo" that should be accepted during future login attempts. An online password cracker will not be able to peek inside the encrypted vault unless he has already guessed the correct password and derived $K_u$. However, the above approach still has a security drawback in that an offline attacker who manages to crack the user's password $\texttt{pwd}_u$ will be able to access *any* password stored in the encrypted vault.

While the TypTop [CWP+17] system maintains a cache of *all* incorrect login attempts, only the passwords that are "sufficiently close" to the original password are considered as candidates to be added to a list of "acceptable typos" for future login attempts. Thus, in the password typo tolerant application, one only needs to store incorrect login attempts that are plausibly typos of the user's original password e.g., the Hamming Distance between the two passwords is at most 2 or the password was incorrectly capitalized because the CAPSLOCK key was not turned off. If the user mistakenly logs into his social media account (12345_SOCIAL) with his bank password (STR@NG_BANK_#;aym7*5) and the social media site was running TypTop then it would add the bank password (STR@NG_BANK_#; aym7*5) to its encrypted cache even though this password is not close to the social media password and would never be added to the list of "acceptable typos". The potential presence of additional user passwords in the encrypted cache could increase the incentive for an offline adversary to crack the social media password (12345_SOCIAL) in order to decrypt the cache which might contain the user's passwords for other accounts e.g., the banking password STR@NG_BANK_#;aym7*5. Ideally, when the authentication server sees an incorrect login attempt it would add the password to the encrypted cache if and only if that login attempt is a plausible typo. However, the authentication server should not store the password $\texttt{pwd}_u$ in plaintext form so it is difficult to know whether or not the login attempt is a plausible typo a priori. To this end it would be useful to generate a "conditional" ciphertext $c'$ such that (1) if $\texttt{pwd}_u$ and $\texttt{pwd}'$ are sufficiently close then $c'$ decrypts to $\texttt{pwd}'$; (2) otherwise $c'$ leaks no information about the password $\texttt{pwd}'$.

## 1.1 Our Contributions

We introduce the notion of a conditional encryption scheme and demonstrate a concrete application to improve the security of personalized typo tolerant systems such as TypTop [CWP+17]. We conjecture that Conditional Encryption may find many other MPC applications e.g., securing Triger-Action Platforms ("If-this-than-that" operations) for IoT services [CCW+21] or designing Fuzzy Password Authenticated Key Exchange Protocols [DHP+18, CHK+05, RX23, BFH+23]. Intuitively, a conditional encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{CEnc})$ for a binary predicate $P(\cdot,\cdot)$ is a public key encryption scheme with the addition of a new "conditional encryption" algorithm. The conditional encryption $\mathsf{CEnc}_{pk}(c, m_2, m_3)$ algorithm accepts four inputs: a public encryption key $pk$, a (regular) public key ciphertext $c = \mathsf{Enc}_{pk}(m_1)$ for an *unknown* message $m_1$, a control message $m_2$ and a payload message $m_3$ and the output is a ciphertext $c'$. Intuitively, if $P(m_1, m_2) = 1$, then control message $m_2$ is related to the unknown and encrypted message $m_1$ (e.g., the Hamming Distance between $m_2$ and $m_1$ is sufficiently small) and the output ciphertext $c'$ should encrypt the payload message $m_3$ i.e., $\mathsf{Dec}_{sk}(c') = m_3$. On the other hand, if $P(m_1, m_2) = 0$ then the messages $m_1$ and $m_2$ are unrelated and the ciphertext $c'$ should not reveal *any* information about the control message $m_2$ or the payload message $m_3$ — even if the attacker knows the secret decryption key $sk$.

### 1.1.1 Conditional Encryption Security

We provide formal security definitions for a conditional encryption scheme in the semi-honest settings. If the attacker does not know the secret decryption key $sk$, then we require that the encryption scheme satisfies the traditional notion of real-or-random security for a public key encryption scheme. When the attacker does have the secret key $sk$ and the predicate does not hold (i.e., $P(m_1, m_2) = 0$) we still want to ensure that the ciphertext $c' = \mathsf{CEnc}_{pk}(c, m_2, m_3)$ does not leak any information about $m_2$, $m_1$ or $m_3$. We formalize this "conditional encryption secrecy" property using a simulator $\mathtt{Sim}(pk)$ who is *only* given access to the public key and must generate a ciphertext which is indistinguishable from $c'$ even if the distinguisher $\mathcal{D}$ is given access to the secret key $sk$ as well as the original ciphertext $c$ and the original messages $m_1$, $m_2$ and $m_3$. We

elect to follow a concrete security definition instead of asymptotic definitions to provide concrete guidance on selecting the concrete security parameters in practice.

### 1.1.2 Efficient Constructions

We next provide *efficient* constructions of conditional encryption for the equality predicate and for predicates based on hamming distance[1], edit distance and CAPSLOCK. Our constructions use the Pallier partially homomorphic encryption scheme, secret sharing and authenticated encryption as the constructive building blocks. We also provide a generic composition theorem for OR predicates in the semi-honest setting. In particular, if we have separate constructions of conditional encryption schemes for predicates $P_1(\cdot,\cdot), \ldots, P_k(\cdot,\cdot)$ then we can obtain a conditional encryption scheme for the predicate $P_{OR}(m_1, m_2) \doteq \bigvee_{i=1}^{k} P_i(m_1, m_2)$ simply by concatenating the conditional ciphertexts generated by the conditional encryption algorithm $\mathsf{CEnc}_i$ for predicate $P_i$. As an application we consider the "typo predicate" which is the OR of several predicates: Hamming Distance at most two, Edit-Distance at most one and a CAPSLOCK predicate. This is the same predicate used by Chaterjee et al. in the TypTop personalized typo tolerant password authentication system. In the appendix, we also provide a general construction of conditional encryption for arbitrary predicates circuit private fully homomorphic encryption (FHE). However, the practicality of this construction is unclear as circuit private FHE is substantially more expensive than Pallier.

### 1.1.3 Application to TypTop

We show how to (slightly) modify the TypTop system to improve security using conditional encryption. In the appendix we formally define the notion of "typo privacy" (see Definition 12) which ensures that the authentication server never collects ciphertexts of passwords which are not plausible typos of the original password. While the original TypTop scheme does not satisfy typo privacy, we prove that our modified TypTop construction does satisfy typo privacy and is still efficient. See Section 3 and Section 4 for more details.

### 1.1.4 Implementation and Empirical Evaluation

We provide a C++ implementation for each of our practical conditional encryption schemes (excluding our general construction from circuit private FHE). Our implementations include conditional encryption scheme for the following predicates: CAPSLOCK, Edit Distance One, Hamming distance at most $t$, as well as OR composition of these predicates (CAPSLOCK or Edit Distance One or Hamming Distance Two). We also modify the C++ implementation of the TypTop system for personalized password typo correction to use conditional encryption for enhanced security (Typo Privacy). We further modified the TypTop system to utilize memory hard functions [AS15, AB17, ABP18, BDK16] for key-derivation — an update recommended by the TypTop authors. Our code is available on Github [AB24a] and Zenodo [AB24b].

We evaluate the performance of our conditional encryption schemes for each predicate. As an example, when we consider the OR predicate for messages of length $n \leq 32$ characters (e.g., almost all passwords[2]) and instantiate the Pallier Cryptosystem with a 1024-bit modulus $N$ we observe average running times 158.15 (ms), 645.945 (ms) and 261.44 (ms) for the regular encryption $\mathsf{Enc}_{pk}(\cdot)$, conditional encryption $\mathsf{CEnc}_{pk}(\cdot)$ and decryption of a conditional ciphertext $\mathsf{CDec}_{sk}(\cdot)$, respectively. The size of a regular (resp. conditional) ciphertext was 16 KB (resp. 24 KB ). The results are summarized in Table 1. We find that modifying the TypTop system does not increase authentication delays although it does increase the storage requirements

[1]For the Hamming Distance predicate (e.g., $P(m_1, m_2) = 1$ if and only if $m_1[i] \neq m_2[i]$ for at most $d$ locations $i \leq n$) decrypting a conditional ciphertext predicate requires time proportional to $\binom{n}{d}$ where $n$ is the length of the messages $m_1$ and $m_2$ and $d$ is the Hamming Distance that we tolerate. This can be slow when both $d$ and $n$ are large. However, in our target password applications the desired distance parameter $d$ for our Hamming Distance predicate (resp. edit-distance predicate) is relatively small (e.g., $d = 2$) as is the parameter $n$ (e.g., $n \leq 32$). Finding efficient constructions for the Hamming Distance (or Edit-Distance) predicate when $n$ and $d$ are both large is left as an open challenge for future research.

[2]Over 99.9% of leaked RockYou passwords were less than 30 characters.

for the authentication server by a factor of $\approx 246$. Fortunately, per user storage will not be a limiting factor in most settings. See Section 4 and Table 2 for more details.

### 1.1.5 Related work

At a high level the notion of conditional encryption might seem similar to other advanced public key primitives such as identity based encryption [BF01, SW05, Gen06, BGK08, BRS13], predicate encryption [KSW08, GVW15b, SSW09, BCFG17, AYY22], attribute based encryption [BSW07, AYY22, Cha07, GPSW06, CC09, LW11, GVW15a, ADMS18, WPC23], functional encryption [NAP+14, GGHZ16a, GGHZ16b, SS10, AGVW13, EM23] and fully homomorphic encryption[Gen09b, BGV14, vGHV10, VJH21]. However, we stress that the security requirements for conditional encryption are quite distinctive in that we require that secrecy guarantees hold *$*even if*$* the attacker has the secret decryption key. By contrast, the security definitions for other public key primitives (identity based encryption, predicate encryption, attribute based encryption, functional encryption and fully homomorphic encryption) all assume that the attacker *does not* have the secret decryption key. While we do use the Paillier partially homomorphic encryption scheme to construct conditional encryption schemes for particular predicates, these constructions do not use Pallier as a blackbox. We are also able to construct conditional encryption for general predicates using *circuit private* FHE, but the circuit privacy requirement seems to be inherent i.e., there exists regular (non circuit private) FHE schemes for which our conditional encryption construction is explicitly broken. See discussion in Appendix D.

## 1.2 Preliminaries

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

Given a randomized algorithm $\mathcal{A}$ (e.g., key-generation) we use $y = A(x; r)$ to denote the deterministic output of $A$ when run on input $x$ with fixed random $r \in \{0,1\}^*$ and we use the random variable $y \leftarrow \mathcal{A}(x)$ to denote the output of $A(x; r)$ when $r$ is selected randomly.

Let $\Sigma$ 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 $i$th character of $w$. We let $\mathcal{M}_n = \Sigma^{\leq n}$ denote the set of all strings $w$ with length $\|w\| \leq n$. It will be convenient to assume that all passwords have the same length. Of course most user passwords do not have the same length but if the maximum length of a user password is $n-1$ then we can easily define a 1 to 1 function $\mathtt{Pad} : \Sigma^{\leq n-1} \to \Sigma^n$ and consider $\mathcal{PWD} = \Sigma^n$ to be the set of all possible user passwords after padding. In practice, we could 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 $x_2$.

Let $L = \langle l_1, \ldots, l_{|L|} \rangle$ be list of $|L|$ elements. We also define the operation $L' = \mathtt{Append}(L, l)$ 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.

For the message $m \in \Sigma^{\leq n}$ we use the notation $m_{-i} \in \Sigma^{\leq n-1}$ to denote the string $m$ when the $i$-th char is deleted and if $i > |m|$ then $m_{-i} = m$.

### 1.2.1 Partially homomorphic Encryption

The Paillier cryptosystem is a partially homomorphic cryptographic scheme which supports ciphertext addition, plaintext to ciphertext multiplication and subtraction. Specifically, the public key $pk = (N, g)$ (resp. secret key $sk = (\beta, \mu)$) consists of $N = pq$ where $p, q$ are prime numbers and the number $g = N + 1 \in \mathbb{Z}^*_{N^2}$ (resp. $\beta = \mathsf{lcm}(p-1, q-1)$ and $\mu = \varphi(N)^{-1} \mod N$). We note that for all $i \in \mathbb{Z}_N$ we have $g^i = \sum_{j=0}^{i} \binom{i}{j} N^j = 1 + Ni \mod N^2$ so that $g$ has multiplicative order $N$ modulo $N^2$. The secret key $sk = (\beta, \mu)$ consists of two parameters $\beta = \mathsf{lcm}(p-1, q-1)$ and $\mu = \varphi(N)^{-1} \mod N$ is defined to be the

multiplicative inverse of $\varphi(N) = (p-1)(q-1)$ modulo $N$.

The algorithm $\mathsf{Enc}_{pk}(m;r)$ takes as input a message $m \in \mathbb{Z}_N$ and a nonce $r \in \mathbb{Z}_N^*$ and outputs $g^m r^N \mod N^2$. The function $\mathsf{Enc}_{pk}$ acts as a bijective map from $\mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$. In particular, for *every* $c \in \mathbb{Z}_{N^2}^*$ there is a message $m \in \mathbb{Z}_N$ and a nonce $r \in \mathbb{Z}_N^*$ such that $c = g^m r^N \mod N^2$ [Pai99].

The encryption scheme has several homomorphic properties in particular if $c_1 = g^{m_1} r_1^N \mod N^2$ and $c_2 = g^{m_2} r_2^N \mod N^2$ encrypt message $m_1, m_2 \in \mathbb{Z}_N$ respectively then $c_1 c_2 = g^{m_1+m_2}(r_1 r_2)^N \mod N^2$ encrypts the message $m_1 + m_2 \mod N$. Similarly, if $c = g^m r^N \mod N^2$ encrypts the message $m$ then $c^k = g^{mk}(r^k)^N \mod N^2$ encrypts the message $mk \mod N$. See Appendix A for a full description of the Paillier encryption scheme.

When we apply the Paillier Cryptosystem, our desired message space $\mathcal{M}$ is typically not the set of integers $\mathbb{Z}_N$. Thus, we assume that there is an injective map $\mathsf{ToInt} : \mathcal{M} \to \mathbb{N}$ and $\mathsf{ToInt}^{-1} : \mathbb{N} \to \mathcal{M}$. We will also assume that $|\mathcal{M}| \leq N$ and that $\forall m \in \mathcal{M}$ that $0 \leq \mathsf{ToInt}(m) < |\mathcal{M}| \leq N$. Given $x \in \mathbb{Z}_N$ we define $\mathsf{ToInt}^{-1}(x) = \perp$ if $x$ has no preimage i.e., $\forall m \in \mathcal{M}$ we have $\mathsf{ToInt}(m) \neq x$.

### 1.2.2 Secret Sharing (SS)

Several of our constructions rely on a primitive called secret sharing. A $(t,n)$-secret sharing scheme consists of two polynomial time algorithms $\mathtt{ShareGen}$ and $\mathsf{SecretRecover}$. Intuitively, $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \mathtt{ShareGen}(n,t,s)$ takes as input a secret $s \in \mathbb{F}$ along with parameters $n, t$ and outputs $n$ shares $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \in \mathbb{F}$. Given any subset $S = \{i_1, \ldots, i_t\} \subseteq [n]$ of $|S| = t$ shares we can recover the secret $s$ using

$$\mathsf{SecretRecover}\left((i_1, \llbracket s \rrbracket_{i_1}), \ldots, (i_t, \llbracket s \rrbracket_{i_t})\right) = s$$

However, given any smaller subset $S = \{i_1, \ldots, i_{t-1}\} \subseteq [n]$ of size $|S| \leq t-1$ shares an attacker cannot infer *anything* about $s$ from the shares $\llbracket s \rrbracket_{i_1}, \ldots, \llbracket s \rrbracket_{i_{t-1}}$. In particular, we require that for all secrets $s \in \mathbb{F}$, all subsets $S = \{i_1, \ldots, i_{t-1}\} \subseteq [n]$ of size $t-1$ the shares $\llbracket s \rrbracket_{i_1}, \ldots, \llbracket s \rrbracket_{i_{t-1}}$ can be viewed as uniformly random independent elements in $\mathbb{F}$ unrelated to the secret $s$. The Shamir Secret sharing scheme [Sha79] satisfies this requirement. See appendix Appendix B for more detail about (Shamir) Secret Sharing.

### 1.2.3 String Distance and Close Passwords

Given a string $w \in \Sigma^n$ and $i \leq n$ we use $w[i] \in \Sigma$ to denote the $i$th character of $\Sigma$ and given two strings $w_1, w_2 \in \Sigma^n$ we use $\mathtt{Ham}(w_1, w_2) = |\{i | w[i] \neq w[j]\}|$ to denote the hamming distance between them. Similarly, given two strings $w_1, w_2 \in \Sigma^*$ we use $\mathtt{ED}(w_1, w_2)$ to denote the edit-distance between them i.e., the minimum number of insertions/deletions to transform $w_1$ into $w_2$ (or vice versa). Note that if $w_1 = w_2$ then $\mathtt{Ham}(w_1, w_2) = 0$ and $\mathtt{ED}(w_1, w_2) = 0$. We will often use Hamming/Edit Distance to determine if two passwords $\mathtt{pwd}_1, \mathtt{pwd}_2$ are close e.g., we could define a predicate $P(\mathtt{pwd}_1, \mathtt{pwd}_2) = 1$ if $\mathtt{Ham}(\mathtt{pwd}_1, \mathtt{pwd}_2) \leq 2$ or $\mathtt{ED}(\mathtt{pwd}_1, \mathtt{pwd}_2) \leq 1$; otherwise $P(\mathtt{pwd}_1, \mathtt{pwd}_2) = 0$. We could also combine Hamming/Edit distance with other common password typos such as CAPSLOCK/SHIFT errors e.g., $P(\mathtt{pwd}_1, \mathtt{pwd}_2) = 1$ if $\mathsf{InvertCase}(\mathtt{pwd}_1) = \mathtt{pwd}_2$ or $\mathtt{Ham}(\mathtt{pwd}_1, \mathtt{pwd}_2) \leq 2$ or $\mathtt{ED}(\mathtt{pwd}_1, \mathtt{pwd}_2) \leq 1$; otherwise, $P(\mathtt{pwd}_1, \mathtt{pwd}_2) = 0$.

## 2 Conditional Encryption

In this section, we will introduce the notion of a *conditional encryption* scheme. A conditional encryption scheme is similar to a regular public key encryption scheme with the addition of a special algorithm $\mathsf{CEnc}_{pk}$. This algorithm takes three 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 $\mathsf{CEnc}_{pk}(c, m_2, m_3) \in \mathcal{C}_{\mathsf{CEnc}}$[3] 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 $sk$.

We now formally define conditional encryption along with its associated security/correctness requirements.

**Definition 1.** *A conditional encryption scheme* $\Pi$ *for a binary predicate* $P : \mathcal{M} \times \mathcal{M} \to \{0,1\}$ *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* $\lambda$ *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* $\mathsf{Enc}_{pk}$*.*
- $(b = 1, \tilde{c}) = \mathsf{CEnc}_{pk}\left((0, c_{m_1}), m_2, m_3; r\right)$*: 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* $\mathsf{CEnc}$ *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* $\mathsf{CEnc}_{pk}\left((1, c_{m_1}), m_2, m_3; r\right) = \perp$ *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

$$\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{Enc}_{pk}\left(m\right)\right) = m\right] = 1$$

i.e., perfect correctness for ciphertexts output by the regular encryption algorithm. For correctness of the conditional encryption algorithm we want to ensure that $\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c', m_2, m_3\right)\right) = m_3$ 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

$$\Pr\left[\mathsf{Dec}_{sk}\left(\mathsf{CEnc}_{pk}\left(c', m_2, m_3\right)\right) = m_3\right] \geq 1 - \epsilon\left(\lambda\right)$$

for a negligible function $\epsilon(\cdot)$ whenever

$$\exists r', m_1 \text{ s.t. } c' = \mathsf{Enc}_{pk}(m_1; r')$$

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. If $c' = (1, \tilde{c})$ was generated by the conditional encryption algorithm then we provide no guarantees that the ciphertext $c'' = \mathsf{CEnc}_{pk}(c', m_2, m_3)$ can be decrypted correctly or is even well formed i.e., in all of our constructions $\mathsf{CEnc}_{pk}(c', m_2, m_3)$ will simply output $\perp$ when $c' = (1, \tilde{c})$ is a conditional ciphertext.

**Definition 2** (Correctness)**.** *We say that* $\Pi$ *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* $\mathsf{Dec}_{sk}\left(\mathsf{Enc}_{pk}\left(m; r_2\right)\right) = m$ *whenever* $\{sk, pk\} \leftarrow \mathsf{KeyGen}(1^\lambda; r_1)$*.*

[3]Similarly, we define $\mathcal{C}_{\mathsf{CEnc}}$ as the ciphertext space for a conditional encryption scheme.

- *(**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*

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

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

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

**Definition 3** (Efficiency)**.** *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* $\lambda$.

We can optionally require that our conditional encryption $\Pi$ is error detecting i.e., whenever $P(m_1, m_2) = 0$, the ciphertext $c = \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ will decrypt to a special symbol $\mathsf{Dec}_{sk}(c) = \perp$ (whp). If $\Pi$ is error detecting this allows us to detect which outputs of the $\mathsf{CEnc}$ algorithm are (in)valid.

**Definition 4** (Error Detecting)**.** *We say that* $\Pi$ *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*

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

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

We now formally define the security of a conditional encryption scheme. Intuitively, in the security game we ask a 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 $c_1$.

**Definition 5** (Conditional Encryption Secrecy)**.** *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* $t(\lambda)$

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

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

The definition of conditional encryption secrecy holds in a semi-honest setting where 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. We remark that this assumption is reasonable in our password typo vault application because the keys and ciphertexts are generated by the authentication server (a trusted party). However, one can imagine applications where we do not want to assume that $c_1$ and $sk$ were generated honestly. We leave it as an open question to define/construct maliciously secure conditional encryption schemes.

**Real-Or-Random Security:** We also require that a conditional encryption scheme satisfies the traditional notion of *real-or-random* (RoR) security i.e., an attacker who does not have the secret key cannot distinguish between real and random ciphertexts. In Appendix C we extend the traditional definition of *real-or-random* (RoR) security to conditional encryption schemes. All of our conditional encryption schemes constructions will satisfy RoR security under the plausible assumption that Pallier encryption itself satisfies RoR security. Because our focus is on conditional encryption secrecy we will defer all RoR security proofs to Appendix C.

# 3 Concrete Constructions of Conditional Encryption

In this section we present concrete constructions of conditional encryption for several different binary predicates. As a warm-up we first consider the equality predicate $P_{=}(m_1, m_2) = 1$ if and only if $m_1 = m_2$. As an application we can use this construction to obtain conditional encryption for the CAPSLOCK predicate since $P_{\mathsf{CAPSLOCK}}(m_1, m_2) = P_{=}(m_1, \mathsf{InvertCase}(m_2))$. We then provide constructions for predicates based on the Hamming Distance (resp. Edit Distance) between $m_1$ and $m_2$. Finally, given conditional encryption schemes for predicates $P_1, \ldots, P_k$ we show how to compose these results to obtain conditional encryption schemes for the OR predicate $P_{OR}(m_1, m_2) \doteq \bigvee_{i=1}^{k} P_i(m_1, m_2)$.

## 3.1 Conditional Encryption for the Equality Test Predicate $P_{=}(x)$

In this part, we will start off by providing a concrete construction of conditional encryption when the predicate is equality test $P_{=}$. That means that given a ciphertext of an unknown message $m_1$, and the input messages $m_2$ and payload $m_3$, we compute the encryption of $m_3$ if and only if $m_1 = m_2$, i.e., $P_{=}(m_1, m_2) = 1$.

Our construction utilizes the Paillier public key encryption scheme (see Appendix A for more details about Pallier) which contains three main algorithms $\Pi_{\mathtt{P}} = (\mathtt{P.KeyGen}, \mathtt{P.Enc}, \mathtt{P.Dec})$. It will also be convenient to let $\mathsf{ToInt}$ denote an injective mapping from our message space $\Sigma^{\leq n}$ to $\mathbb{Z}_{|\Sigma|^{n+1}}$ and use $\mathsf{ToInt}^{-1}$ to denote the inverse mapping — we define $\mathsf{ToInt}^{-1}(y) = \perp$ if there is no preimage $m \in \Sigma^{\leq n}$ such that $y = \mathsf{ToInt}(m)$.

Our conditional encryption construction $\Pi$ sets $\Pi.\mathsf{KeyGen} \doteq \mathtt{P.KeyGen}$ and we set $\Pi.\mathsf{Enc}_{pk}(m_1) \doteq (0, \mathtt{P.Enc}_{pk}(m_1))$ i.e., to encrypt $m_1 \in \Sigma^{\leq n}$ we simply compute $c = \mathtt{P.Enc}_{pk}(\mathsf{ToInt}(m_1))$ and output the ciphertext $c_{m_1} = (b = 0, c)$. Given a ciphertext $c = (0, c_1)$ with flag $b = 0$ we define $\Pi.\mathsf{Dec}_{sk}(c = (0, c_1)) \doteq \mathsf{ToInt}^{-1}(\mathtt{P.Dec}_{sk}(c_1))$. The conditional encryption algorithm $\Pi.\mathsf{CEnc}_{pk}(c_{m_1} = (0, c), m_2, m_3)$ works as follows: First we extract $N$ from the public key $pk$ and compute $\hat{m}_2 = \mathsf{ToInt}(m_2)$ and $\hat{m}_3 = \mathsf{ToInt}(m_3)$ to map them to Paillier's plaintext space. Next we pick random numbers $R \in_R \mathbb{Z}_N$ and $r \in_R \mathbb{Z}_N^*$ and compute $c^{=} = c^R (N+1)^{-R\hat{m}_2 + \hat{m}_3} r^N \mod N^2$. Finally, we output $\Pi.\mathsf{CEnc}_{pk}(c_{m_1} = (0, c), m_2, m_3; R, r) = (1, c^{=})$. Given a conditional ciphertext $(1, c^{=})$ the decryption algorithm will simply output $\Pi.\mathsf{Dec}_{sk}(1, c^{=}) \doteq \mathsf{ToInt}^{-1}(\mathtt{P.Dec}_{sk}(c^{=}))$. See **Construction** 18 in Appendix I for a more formal description of our construction.

Intuitively, we have $c = (N+1)^{\hat{m}_1} r_1^N \mod N^2$ for some message $m_1 = \mathsf{ToInt}^{-1}(\hat{m}_1)$ and random value $r_1 \in \mathbb{Z}_N^*$. In this case, the final ciphertext $c^{=}$ can be written as $c^{=} = (N+1)^{R(\hat{m}_1 - \hat{m}_2) + \hat{m}_3} (r r_1^R)^N \mod N^2$. If $\hat{m}_1 - \hat{m}_2 = 0$ then $R(\hat{m}_1 - \hat{m}_2) = 0$ cancels and we are left with a valid encryption of $m_3$. Otherwise, the value $R(\hat{m}_1 - \hat{m}_2) + \hat{m}_3 \mod N$ can be viewed as a fresh Paillier ciphertext encrypting a uniformly random integer in $\mathbb{Z}_N$. More specifically, let $y = \mathsf{ToInt}(m_2) - \mathsf{ToInt}(m_1)$. As long as $\gcd(N, y) = 1$ (due to the selection of $p, q$ s.t., $\min(p, q) > |\Sigma^{n+1}|$) the value $R' = R(\hat{m}_1 - \hat{m}_2) + \hat{m}_3 \mod N$ will also be distributed uniformly in $\mathbb{Z}_N$ i.e., for all $x \in \mathbb{Z}_N$ we have $\Pr[yR = x \mod N] = \Pr[R = xy^{-1} \mod N] = 1/N$ when $R \in \mathbb{Z}_N$ is random and $y^{-1}$ is the multiplicative inverse of $y \mod N$. When we pick our Paillier public key ,we can ensure we always have $\gcd(N, y) = 1$ by selecting primes $p$ and $q$ such that $p, q \geq \max_{m_1 \in \Sigma^n} \mathsf{ToInt}(m_1) < |\Sigma^{n+1}|$ and setting $N = pq$.

**Theorem 1.** ***Construction** 18 is a perfectly correct and $1 - \epsilon(\lambda)$-error detecting conditional encryption scheme with $\epsilon(\lambda) = \dfrac{|\Sigma|^{n+1}}{N} \leq \dfrac{1}{\max\{p, q\}}$.*

The proof of Theorem 1 can be found in Appendix E.

**Theorem 2.** *The conditional encryption scheme described in **Construction** 18 provides $(\infty, t_{\mathtt{Sim}}, 0)$ conditional encryption secrecy in which $t_{\mathtt{Sim}} = t_{\mathtt{P.Enc}}$ is time of doing one Paillier Encryption.*

Intuitively, the simulator $\mathtt{Sim}(pk)$ simply picks random values $r' \in \mathbb{Z}_N^*$ and $R' \in \mathbb{Z}_N$ and outputs $(N+1)^{R'} r'^N \mod N^2$ i.e., the Paillier encryption of a uniformly random message. Intuitively, if $\hat{m}_1 \neq \hat{m}_2$ and $\mathbf{gcd}(\hat{m}_1 - \hat{m}_2, N) = 1$ then the value $R(\hat{m}_1 \neq \hat{m}_2) + \hat{m}_3$ is uniformly random in $\mathbb{Z}_N$ i.e., statically indistinguishable from $R'$. Similarly, if we fix any $z \in \mathbb{Z}_N^*$ (we will use $z = r_1^R \mod N$) and pick $r \in \mathbb{Z}_N^*$ randomly then the value $rz \mod N$ is also uniformly random in $\mathbb{Z}_N^*$ i.e., statically indistinguishable from $r'$.

The formal proof of Theorem 2 is available in Appendix E. We also prove that **Construction** 18 satisfies the traditional notion of Real-or-Random security — see Theorem 9 in Appendix C.

### 3.2 CAPSLOCK Predicate

We can immediately use our conditional encryption scheme for equality test to obtain a construction for the the CAPSLOCK predicate $P_{\mathsf{CAPSLOCK}}$ which is defined as $P_{\mathsf{CAPSLOCK}}(m_1, m_2) = 1$ if and only if $m_1 = \mathsf{InvertCase}(m_2)$; otherwise $P_{\mathsf{CAPSLOCK}}(m_1, m_2) = 0$. Observe that we can equivalently define

$$P_{\mathsf{CAPSLOCK}}(m_1, m_2) = P_{=}(m_1, \mathsf{InvertCase}(m_2)).$$

Thus, our conditional encryption construction $\Pi_{\mathsf{CAPSLOCK}}$ for $P_{\mathsf{CAPSLOCK}}$ is exactly as our construction $\Pi_{=}$ for $P_{=}$ with the following modification to the conditional encryption algorithm $\Pi_{\mathsf{CAPSLOCK}}.\mathsf{CEnc}_{pk}(c_{m_1}, m_2, m_3) = \Pi_{=}.\mathsf{CEnc}_{pk}(c_{m_1}, \mathsf{InvertCase}(m_2), m_3)$ Conditional encryption secrecy and correctness of the construction $\Pi_{CPSLCK}$ follows immediately from Theorem 2 and Theorem 1 respectively. RoR security also follows directly from RoR security of **Construction** 18 — see Theorem 9 in Appendix C for RoR security.

### 3.3 Hamming Distance Predicate

We now describe our conditional encryption construction for the Hamming Distance predicate $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 1$ if and only if $\mathtt{Ham}(m_1, m_2) \leq \ell$. For ease of exposition, we will describe our construction under the assumption that all messages $m_1, m_2 \in \Sigma^n$ have the exact same length. In several of our applications (e.g., password typos) it may not necessarily be the case that all messages have the same length. However, we can easily deal with this issue by defining an injective padding function $\mathtt{Pad} : \Sigma^{\leq m} \to \Sigma'^n$ where $\Sigma' = \Sigma \cup \{y\}$ extends the alphabet $\Sigma$ by adding a new symbol $y$. In particular, given $m \in \Sigma^{\leq n}$ we define $\mathtt{Pad}(m) = m \| y^{n-|m|}$ to be the length $n$ string from our larger alphabet $\Sigma'$ obtained by padding $m$ with $y$'s. Clearly, the function $\mathtt{Pad}$ is injective so we can define an inverse $\mathtt{Pad}^{-1}$ such that $\mathtt{Pad}^{-1}(\mathtt{Pad}(m)) = m$ for any $m \in \Sigma^{\leq m}$. Given two messages $m, m' \in \Sigma^{\leq n}$ and an integer $\ell \geq 0$, we define the binary predicate $P_{\ell,\mathtt{Ham},\mathtt{Pad}}(m, m') = 1$ if and only if $\mathtt{Ham}(\mathtt{Pad}(m), \mathtt{Pad}(m')) \leq \ell$. Clearly, if we have a conditional encryption scheme for the Hamming Distance predicate $P_{\ell,\mathtt{Ham}}(m_1, m_2)$ with message space $m_1, m_2 \in \Sigma'^n$ then we can immediately apply padding to obtain a conditional encryption scheme for the related predicate $P_{\ell,\mathtt{Ham},\mathtt{Pad}}(m_1, m_2)$ with message space $m_1, m_2 \in \Sigma^{\leq n}$.

**Attempt 1:** As an initial attempt at constructing conditional encryption for our predicate $P_{\ell,\mathtt{Ham}}$ we can use our equality predicate construction as a blackbox along with the Shamir secret sharing scheme $\mathsf{SS} = (\mathsf{ShareGen}, \mathsf{SecretRecover})$ and a symmetric key authenticated encryption scheme. The basic idea is to split messages into individual characters $m_1 = (m_1[1], \ldots, m_1[n])$ and encrypt character by character to obtain $c_1 = (c_1[1], \ldots, c_1[n]) = \mathsf{Enc}_{pk}(m_1)$. The conditional encryption algorithm $\mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ would similarly split the control message $m_2$ up into $n$ individual characters and generate $n$ secret shares $[\![s]\!]_1, \ldots, [\![s]\!]_n$ of a fresh symmetric key $K$. We would then use the original conditional encryption scheme for $P_{=}$ to compute $c_2[i] = \Pi_{=}.\mathsf{CEnc}_{pk}(c_i, m_2[i], [\![s]\!]_i)$ for each $i \leq n$ using $m_2[i]$ as the control message and $[\![s]\!]_i$ as the payload message. The final conditional ciphertext would include $c_2[1], \ldots, c_2[n]$ as well as an encryption of $m_3$ using the symmetric key $K$. The decryption algorithm would decrypt each $c_2[i]$ to obtain the share $[\![s]\!]_i$. As long as the Hamming Distance predicate holds the decryption algorithm would obtain enough shares to recover $K$ and decrypt $m_3$.

The problem with this construction is that if the predicate does not hold then an attacker who knows the secret key can still (whp) identify which shares are valid. In particular, it would be trivial for a party who knows the secret key $sk$ to distinguish between the encryption of a valid share $[\![s]\!]_i < 2^\lambda$ (recovered when $m_1[i] = m_2[i]$) and the encryption of a random element in $\mathbb{Z}_N$ (as is the case when $m_2[i] \neq m_1[i]$) since $2^\lambda \ll N$. This would allow the attacker to learn the set $S = \{i : m_1[i] = m_2[i]\}$ of indices $i \leq n$ where $m_2$ matches $m_1$ even if $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 0$ — a clear violation of conditional encryption secrecy!

**The Fix:** To address the above issue we use a randomized encoding to ensure that, when the predicate does not hold, it is impossible to identify which shares are (in)valid. In particular, instead of computing

$c_2[i] = \Pi_=.\mathsf{CEnc}_{pk}(c_i, m_2[i], [\![s]\!]_i)$ we instead compute $c_2[i] = \Pi_=.\mathsf{CEnc}_{pk}(c_i, m_2[i], x_i)$ where $x_i = \mathsf{REnc}([\![s]\!]_i)$ is a random encoding of the share $[\![s]\!]_i$ as an integer in larger Pallier plaintext space $\mathbb{Z}_N$. In more detail $\mathsf{REnc}(x) = a_i 2^\lambda + x$ where the value $a_i \leq \lfloor \frac{N-1-x}{2^\lambda} \rfloor$ is chosen uniformly at random. Intuitively, $x_i = \mathsf{REnc}([\![s]\!]_i)$ encodes the share $[\![s]\!]_i$ as a random element in $\mathbb{Z}_{N'}$ subject to the constraint that $[\![s]\!]_i = \mathsf{REnc}([\![s]\!]_i) \mod 2^\lambda$. Since the value of the share $[\![s]\!]_i$ itself is random this ensures that the attacker cannot distinguish $x_i$ from a random element in $\mathbb{Z}_N$ — unless the predicate $P_{\ell,\mathtt{Ham}}$ holds and we can recover enough correct shares to recover the secret decyrption key. Decrypting a conditional ciphertext will require more work since we do not know a priori which recovered shares are valid and we have to consider all possible subsets. Fortunately, when $\ell$ is constant the number of subsets remains polynomial in $n$.

In a bit more detail the conditional encryption scheme $\Pi$ works as follows:

(1) The regular encryption algorithm $\Pi.\mathsf{Enc}_{pk}(m)$ takes as input a message $m = (m[1], \ldots, m[n]) \in \Sigma^n$ and encrypts $m$ character by character to obtain a vector of Paillier ciphertexts $c = (c[1], \ldots, c[n])$ where $c_i = \mathtt{P}.\mathsf{Enc}_{pk}(\mathsf{ToInt}(m[i]); r_i)$ The regular encryption algorithm outputs $(b = 0, c[1], \ldots, c[n])$ where the flag $b = 0$ indicates that this ciphertext was produced by the regular encryption algorithm.

(2) The conditional encryption algorithm $\Pi.\mathsf{CEnc}(c_1, m_2, m_3)$ takes as input 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 $m_3$. We first generate a random symmetric key $K \in \{0,1\}^\lambda$ for our authenticated encryption scheme and encrypt the payload message $m_3$ using $K$ to obtain $c_{AE} = \mathsf{AuthEnc}_k(m_3)$. Second we use the Shamir secret sharing scheme to generate $n$ shares $([\![s]\!]_1, \ldots, [\![s]\!]_n) \leftarrow \mathsf{ShareGen}(n, n-\ell, K)$ for our secret key $K$. We configure our secret sharing scheme such that $n - \ell$ shares are sufficient to recover $K$, but any subset of $n - \ell - 1$ shares information theoretically leaks nothing about $K$. We now follow our equality test construction and compute $c[i] = c_1[i]^{R_i}(N+1)^{-R_i m_2[i] + x_i} r_i^N$ where $x_i = \mathsf{REnc}([\![s]\!]_i)$ is the random encoding of the share $[\![s]\!]_i$, $R_i$ is a uniformly random integer in $\mathbb{Z}_N$ and $r_i$ is uniformly random in $\mathbb{Z}_N^*$. Our final output is $(b = 1, c, c_{AE})$ in which $c = (c[1], \ldots, c[n])$.

(3) 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 $m[i] = \mathsf{ToInt}(x_i)$ and $x_i = \mathtt{P}.\mathsf{Dec}_{sk}(c[i])$. Given a conditionally encrypted ciphertext $(b = 1, c[1], \ldots, c[n], c_{AE})$ we will first extract shares $[\![s']\!]_i = \mathsf{RDec}(x_i)$ with $x_i = \mathtt{P}.\mathsf{Dec}_{sk}(c[i])$. We will then look through all $\binom{n}{n-\ell}$ subsets $S \subseteq [n]$ of $n - \ell$ indexes and their corresponding shares to recover a string

$$K_S = \mathsf{SecretRecover}\left(\left(S[i], [\![s']\!]_{S[i]}\right) \forall 0 \leq i \leq n - \ell\right)$$

which may or may not be valid. If $\mathsf{Auth.Dec}_{K_S}(c_{AE}) = \perp$ then we conclude that $K_S$ is invalid and move on to the next subset; otherwise if $\mathsf{Auth.Dec}_{K_S}(c_{AE}) = m$, we return $m$. If $\mathsf{Auth.Dec}_{K_S}(c_{AE}) = \perp$ for all subsets $S \subseteq [n]$ and their $n - \ell$ corresponding shares, then we output $\perp$.

See **Construction** 19 in Appendix I for a formal description of the construction and see Theorem 15 in Appendix E for a proof that **Construction** 19 is $1 - \epsilon(\lambda)$-correct and a $1 - \epsilon(\lambda)$-error detecting for a negligible function $\epsilon(\lambda)$.

### 3.3.1 Correctness of the Construction 19

We now prove that **Construction** 19, satisfies the security definition Definition 5, i.e., conditional encryption secrecy. We first make a basic statistical observation.

**Theorem 3.** *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 $\mathcal{D}_{ak}$ which outputs random values between $0, \cdots, ak$. Then the statistical distance between these two distributions is $\mathtt{SD}(\mathcal{D}_{ak}, \mathcal{U}_b) = \frac{r}{b} \leq \frac{1}{k+1}$.*

Intuitively, Theorem 3 implies that we cannot distinguish between $\mathsf{REnc}(x)$ and a uniformly random $y \in \mathbb{Z}_N$ whenever $0 \leq x < 2^\lambda$ is picked randomly. The proof of Theorem 3 can be found in Appendix E.

**Theorem 4.** *[Conditional Encryption Secrecy of **Construction** 19] 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 $\lambda$ and any running time parameter $t_{AE}$. Then for any $t$ and any security parameter $\lambda$ **Construction** 19 provides $(t, t_{\mathtt{Sim}}, \epsilon(t, \lambda))$ conditional encryption secrecy with $\epsilon(t, \lambda) \leq \epsilon_{AE}(t, \lambda) + 2^{-\lambda}$ and $t_{\mathtt{Sim}} = n \cdot t_{P.\mathsf{Enc}} + \mathtt{poly}(\lambda)$.*

Theorem 4 follows by applying Theorem 3 with $b = N$, $a = \lfloor N/2^\lambda \rfloor$ and $k = 2^\lambda$. We defer the formal proof of Theorem 4 and Theorem 3 to Appendix E where we also prove that **Construction** 19 provides Real-or-Random security (see Theorem 10).

### 3.3.2 Efficiency

The running time of the key generation algorithm $\mathsf{KeyGen}$ is essentially equivalent to Pallier — with high probability we will have $\min\{p, q\} > \max\left\{2n2^{2\lambda}, |\Sigma|\right\}$. The running encryption algorithm $\mathsf{Enc}$ is essentially $n \times t_p$ where $t_p$ denotes the running time for regular Pallier Encryption and the resulting ciphertext has size $1 + n \cdot \lceil \log_2 N^2 \rceil$ (bits). The running time for the conditional encryption algorithm is essentially $n \times t_p + t_{AE} + t_{SS}$ where $t_p$ (resp. $t_{AE}$, $t_{SS}$) denotes the time for one Pallier Encryption (resp. one authenticated encryption/one execution of $\mathsf{ShareGen}$ over a field of size $2^\lambda$). The size of a conditionaly encrypted ciphertext is $1+n\lceil \log_2 N^2 \rceil + s_{AE}$ where $s_{AE}$ denotes the length of the authenticated encryption ciphertext. The running time of $\mathsf{Dec}_{sk}$ on a conditionally encypted ciphertext is roughly $\binom{n}{\ell}(t_{SSrec} + t_{AE})$ where $t_{SSrec}$ (resp. $t_{AE}$) denotes the running time for $\mathsf{SecretRecover}$ over a field of size $2^\lambda$ (resp. $\mathsf{Auth.Dec}$). If we incorporate a second secret sharing scheme over a smaller finite field then it is possible to slightly optimize the performance to achieve running time $\binom{n}{\ell}t'_{SSrec} + O(t_{SSrec} + t_{AE})$ where $t'_{SSRec}$ denotes the execution time for secret share recovery over the *smaller* finite field — see details in Section 5.1.2.

## 3.4 Edit Distance One

Given two messages $m, m' \in \Sigma^{\leq n}$ and an integer $\ell \geq 0$, we define the binary predicate $P_{\ell,\mathtt{ED}}(m, m') = 1$ if and only if $\mathtt{ED}(m, m') \leq \ell$; otherwise, $P_{\ell,\mathtt{ED}}(m, m') = 0$. In this section, we will construct a conditional encryption scheme for $P_{1,\mathtt{ED}}$ i.e., edit-distance 1. It would be possible to implement the same general construction for $\ell > 1$. However, the ciphertext sizes would grow proportional to $O\left(n^\ell\right)$. Thus, we focus on the $\ell = 1$ case since it is the most useful case for password typo correction (and yields the most efficient construction). We will let $\Pi_{1,\mathtt{ED}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$ denote our conditional encryption for $P_{1,\mathtt{ED}}$ described below.

Given $m = (m[1], \ldots, m[k]) \in \Sigma^k$ we define

$$m_{-i} = (m[1], \ldots, m[i-1], m[i+1], \ldots, m[k]) \in \Sigma^{k-1}$$

to be the string obtained by deleting the $i$th character from $m$ e.g., if $m =$"bead" then $m_{-2}=$"bad". If $j = 0$ or $j > k = |m|$ then we just define $m_{-j} = m$. Observe that $P_{1,\mathtt{ED}}(m, m') = 1$ if and only if there exists $j$ such that $m_{-j} = m'$ or such that $m = m'_{-j}$.

With this observation our construction for $P_{1,\mathtt{ED}}$ will use our construction for $P_=$ as a black box. $\mathsf{KeyGen}(1^\lambda)$ works in the exact same way as the conditional encryption scheme for the equality predicate and will generate a key $(sk, pk = (N, g = N + 1))$ with $N = pq$ and $\min\{p, q\} \geq |\Sigma|^{n+1}$. Our regular encryption algorithm $\mathsf{Enc}_{pk}(m)$ takes as input $m \in \Sigma^{\leq n}$ and outputs a vector $(0, \tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_n)$ where $\tilde{c}_i = P.\mathsf{Enc}_{pk}(\mathsf{ToInt}(m_{-i}))$is the Pallier encryption of $m_{-i}$ encoded as an integer using the injective mapping $\mathsf{ToInt} : \Sigma^{\leq n} \to \mathbb{Z}_{|\Sigma|^{n+1}}$ — $\mathsf{ToInt}^{-1}$ is the inverse mapping. The conditional encryption algorithm $\mathsf{CEnc}_{pk}(c_1, m', m'')$ works by running $\Pi_=.\mathsf{CEnc}_{pk}$, the conditional encryption algorithm for the equality predicate, on $2n + 1$ different inputs to generate $\tilde{c}_0, \tilde{c}_1, \ldots, \tilde{c}_{2n}$ — if for some $j$ we have $\tilde{c}_j = \perp$ then we simply output $\perp$. First, we parse $c_1 = (0, c_1[0], \ldots, c_1[n])$ and set $\tilde{c}_i = \Pi_=.\mathsf{CEnc}_{pk}(c_1[i], m', m'')$ for each $0 \leq i \leq n$. Intuitively, if $m' = m_{-i}$ then $\tilde{c}_i$ is a Pallier encryption of our payload $m''$; otherwise, $\tilde{c}_i = g^{y_i} r_i^N \mod N^2$ will be the random Pallier encryption of a uniformly random $y_i \in \mathbb{Z}_N$ under a uniformly random nonce $r_i \in \mathbb{Z}_N^*$.

Similarly, we can set $\tilde{c}_{n+i} = \Pi_{=}.\mathsf{CEnc}_{pk}(c_1[0], m'_{-i}, m'')$ for each $1 \leq i \leq n$. Intuitively, if $m'_{-i} = m_{-0} = m$ then $\tilde{c}_{n+i}$ is a Pallier encryption of our payload $m''$; otherwise, $\tilde{c}_i = g^{y_i} r_i^N \mod N^2$ will be the random Pallier encryption of a uniformly random $y_i \in \mathbb{Z}_N$ under a uniformly random nonce $r_i \in \mathbb{Z}_N^*$. The decryption algorithm $\mathsf{Dec}_{sk}$ is defined in the natural way. In particular, $\mathsf{Dec}_{sk}(0, c[0], c[1], \ldots, c[n])$ simply decrypts $c[0]$ as $x_0 = P.\mathsf{Dec}_{sk}(c[0])$ using regular Pallier decryption $P.\mathsf{Dec}$ and then outputs $m = \mathsf{ToInt}^{-1}(x_0)$.

Similarly, $\mathsf{Dec}_{sk}(1, \tilde{c}_0, \ldots, \tilde{c}_{2n})$ will run our conditional decryption algorithm $P_{=}$ on each individual ciphertext $\tilde{c}_i$ to recover $x_0, x_1, \ldots, x_{2n}$ with $x_i = P.\mathsf{Dec}_{sk}(\tilde{c}_i)$. If $\min\{x_0, \ldots, x_{2n}\} > |\Sigma|^{n+1}$ then we output $\perp$; otherwise we can simply return $\mathsf{ToInt}^{-1}(\min\{x_0, \ldots, x_{2n}\})$

**Theorem 5.** *$\Pi_{1,\mathtt{ED}}$ is a $1 - \epsilon(\lambda)$ correct conditional encryption scheme for the predicate $P_{1,\mathtt{ED}}$ and $\Pi_{1,\mathtt{ED}}$ is $1 - \epsilon(\lambda)$-error detecting with $\epsilon(\lambda) = \frac{(2n+1)|\Sigma|^{n+1}}{N} \leq \frac{2n+1}{\max\{p,q\}}$.*

**Theorem 6.** *$\Pi_{1,\mathtt{ED}}$ provides $(\infty, t_{\mathtt{Sim}}, 0)$ conditional encryption secrecy for the predicate $P_{1,\mathtt{ED}}$. Here, $t_{\mathtt{Sim}} = (2n+1)t_{\mathtt{P.Enc}}$ is time of doing $(2n+1)$ Paillier Encryptions.*

**Proof of Theorem 6:** (Sketch) Assume that $P_{1,\mathtt{ED}}(m, m') = 0$ it follows from Theorem 2 that for any $j \leq n$ that $\mathsf{CEnc}^{=}_{pk}(\mathsf{Enc}_{pk}(m_{-j}), m', m'')$ outputs $g^{R_j} r_j^N \mod N^2$ for a uniformly random $R_j \in \mathbb{Z}_N$ and $r_j \in \mathbb{Z}_N$. Similarly, for any $j \leq n$ it follows that $\mathsf{CEnc}^{=}(\mathsf{Enc}_{pk}(m), m'_{-j}, m''))$ outputs $g^{R_j} r_j^N \mod N^2$ for a uniformly random $R_j \in \mathbb{Z}_N$ and $r_j \in \mathbb{Z}_N$. Thus, $\mathsf{CEnc}_{pk}(\mathsf{Enc}_{sk}(m_{-j}, m', m''))$ outputs $(1, \tilde{c}_0, \ldots, \tilde{c}_{2n})$ where for each $j \leq 2n$ the Pallier Ciphertext $\tilde{c}_i$ is uniformly random in $\mathbb{Z}_{N^2}^*$.

We define the simulator $\mathtt{Sim}(pk)$ as follows. The simulator $\mathtt{Sim}(pk)$ takes as input the Paillier public key $pk$. For each $0 \leq i \leq 2n+1$ the simulator then selects $R_i \in_R \mathbb{Z}_N$ and $r_i \in_R \mathbb{Z}_N^*$ uniformly at random and then encrypts $R_s$ as $C_{\mathtt{Sim}}[i] = \mathtt{P}.\mathsf{Enc}_{pk}(R_i; r_i) = g^{R_i} r_i^N \mod N^2$ i.e., $C_{\mathtt{Sim}}[i]$ is uniformly random in $\mathbb{Z}_{N^2}^*$. Finally, the simulator outputs $C_{\mathtt{Sim}} = (1, C_{\mathtt{Sim},0}, \ldots, C_{\mathtt{Sim},2n})$. □

## 3.5 OR Composition

Suppose we have conditional encryption schemes $\Pi_1, \ldots, \Pi_k$ for $k$ different predicates $P_1, \ldots, P_k$ and that each scheme has the same message space. Let $P_{or}(m_1, m_2) = \bigvee_{i=1}^k P_i(m_1, m_2)$ the predicate which is 0 (false) if and only if all of the predicates are false i.e., $P_i(m_1, m_2) = 0$ for all $i \leq k$. We will define a conditional encryption scheme $\Pi_{or} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$ for the predicate $P_{or}$.

Intuitively, our key generation algorithm $\mathsf{KeyGen}(1^\lambda)$ runs $(sk_i, pk_i) \leftarrow \mathsf{KeyGen}_i(1^\lambda)$ for each $i$ and outputs $(sk, pk)$ where $sk = (sk_1, \ldots, sk_k)$ and $pk = (pk_1, \ldots, pk_k)$[4]. The algorithm $\mathsf{Enc}_{pk}(m)$ simply generates $c_i = \Pi_i.\mathsf{Enc}_{pk}(m)$ for each $i \leq k$ and outputs $(0, c_1, \ldots, c_k)$. Similarly, the algorithm $\mathsf{CEnc}_{pk}(c = (0, c_1, \ldots, c_k), m', m'')$ simply generates $c_i = \Pi_i.\mathsf{CEnc}_{pk}(c_i, m', m'')$ for each $i \leq k$ and outputs $(0, \tilde{c}_1, \ldots, \tilde{c}_k)$ — if $\tilde{c}_i = \perp$ for any $i \leq k$ then we instead output $\Pi_{or}.\mathsf{CEnc}_{pk}(c, m', m'') = \perp$. Finally, the $\mathsf{Dec}_{sk}(c)$ will run $m_i = \Pi_i.\mathsf{Dec}_{sk}(c)$ to obtain $m_i \in \mathcal{M} \cup \{\perp\}$. If $m_i = \perp$ for all $i \leq k$ then the algorithm outputs $\perp$; otherwise we output $m_j$ where $j$ is largest integer such that $m_j \neq \perp$.

**Theorem 7.** *Suppose that we are given $k$ separate conditional encryption schemes $\Pi_1, \ldots, \Pi_k$ corresponding predicates $P_1, \ldots, P_k$ and that each $\Pi_i$ provides $(t(\lambda), t_{\mathtt{Sim},i}(\lambda), \epsilon_i(t(\lambda), \lambda))$-conditional encryption secrecy. The construction $\Pi_{or}$ provides $(t'(\lambda), t'_{\mathtt{Sim}}(\lambda), \epsilon'(t'(\lambda), \lambda))$-conditional encryption secrecy with $t'(\lambda) = O(t(\lambda))$, $t'_{\mathtt{Sim}}(\lambda) \approx \sum_{i=1}^k t_{\mathtt{Sim}_i}(\lambda)$ and $\epsilon'(t'(\lambda), \lambda) = \sum_i^k \epsilon_i(t'(\lambda), \lambda)$.*

The formal proof is available in Appendix E. Intuitively, the simulator $\mathtt{Sim}_{OR}(pk)$ for $\Pi_{OR}$ will run the simulator $\mathtt{Sim}_i(pk_i)$ for each conditional encryption scheme and concatenate all of the ciphertexts.

**Theorem 8.** *Suppose that we are given $k$ separate conditional encryption schemes $\Pi_1, \ldots, \Pi_k$ corresponding to predicates $P_1, \ldots, P_k$ and that each $\Pi_i$ is $1 - \epsilon_i(\lambda)$-correct and $1 - \epsilon'_i(\lambda)$-error detecting. Then the construction $\Pi_{or}$ is $1 - \epsilon(\lambda)$-correct (resp.$1 - \epsilon'(\lambda)$-error detecting) with $\epsilon(\lambda) = \sum_i^k \epsilon'_i(\lambda) + \sum_i^k \epsilon_i(\lambda)$ (resp. $\epsilon'(\lambda) = \sum_{i=1}^\lambda \epsilon'_i(\lambda)$).*

[4] As an optimization if $\Pi_i.\mathsf{KeyGen}_i(1^\lambda)$ generates a Pallier key for each $i$ then we can generate one Pallier key $(sk_0, pk_0)$ and set $(sk_i, pk_i) = (sk_0, pk_0)$ for all $1 \leq i \leq k$.

The formal proof is available in Appendix E. We also prove that the suggested construction provides Real-or-Random security as well — see Theorem 11 and its corresponding proof in Appendix C.

# 4 The Typo Predicate: Personalized Typo Correction

Motivated by the application of password typo correction we now introduce the predicate $P_{typo}(m_1, m_2) = P_{\mathsf{CAPSLOCK}}(m_1, m_2) \vee P_{\ell=2,\mathtt{Ham},\mathtt{Pad}} \vee P_{\ell=1,\mathtt{ED}}$. Chaterjee et al. [CWP+17] conducted an empirical study of password typos finding that nearly 78% of legitimate typos fit one of the above three categories i.e., CAPSLOCK error, Hamming Distance $\leq 2$ or a single character insertion/deletion. As application of Theorem 7 we obtain a conditional encryption scheme $\Pi_{typo}$ for the predicate $P_{typo}$ with $(t(\lambda), t_{sim}, \epsilon(t(\lambda), \lambda))$-security for $\epsilon(t(\lambda), \lambda) = 2^{-\lambda} + \epsilon_{AE}(t(\lambda), \lambda)$. Correctness and error detection of $\Pi_{typo}$ follow directly from Theorem 8.

### 4.0.1 Application to Personalized Password Typo Correction

We can use our conditional encryption scheme to fix a drawback in the personalized typo correction scheme of Chaterjee et al. [CWP+17].

### 4.0.2 The Security Issue.

Chaterjee et al. [CWP+17] proposed to derive a public/secret key pair $(pk_u, sk_u)$ for every user $u$. The public key $pk_u$ is stored on the authentication server. Any incorrect login attempt $pw' \neq pwd_u$ for this account is encrypted $C_{pw} = \mathsf{Enc}_{pk}(pw)$ and stored in a typo vault. The secret key $sk_u$ is not directly stored on the server, but can be recovered whenever the user logs in with the correct password $pw_u$. In particular, we store $c_{sk_u} = \mathsf{Auth.Enc}_{K_u}(sk_u)$ where the symmetric key $K_u = \mathtt{KDF}(s_u, pwd_u)$ is derived from the users password $pwd_u$ and a random salt value $s_u$ that is stored on the server in plaintext form. Thus, once the correct password $pw_u$ it is possible to recover $K_u$, then $sk_u$, decrypt all of the password in the vault and identify common typos. The drawback of this approach is that *every* incorrect login attempt will appear in the encrypted typo vault. It is not unlikely that the typo vault might include unrelated passwords from the user's other accounts. This could significantly increase the incentives for a rational offline brute-force attacker to crack the user's password [BHZ18], and simultanously increasing the potential harm to users.

### 4.0.3 The Fix

Our fix is straightforward: replace the regular encryption scheme with our conditional encryption scheme for the predicate $P_{typo}$! In addition to $pk_u$ the authentication server will also store $c_u = \mathsf{Enc}_{pk_u}(pwd_u)$ the encryption of the user's password under $pk_u$. Now whenever their is an incorrect login attempt $pw' \neq pwd_u$ we can set $c_{pw'} = \Pi_{typo}.\mathsf{CEnc}_{pk_u}(c_u, pw', pw')$. If $P_{typo}(pwd_u, pw') = 1$ then we have $\Pi_{typo}.\mathsf{Dec}_{sk_u}(c_{pw'}) = pw'$ so that we can recover $pw'$ later when the user logs into the server with the correct password. However, if $P_{typo}(pwd_u, pw') = 0$ then the ciphertext $c_{pw'}$ will be *entirely* useless to an offline attacker even if the attacker can recover $pwd_u$ and $sk_u$!

### 4.0.4 Security Proof

In Appendix G we formalize the notion of typo privacy (see Definition 12) for an authentication server that maintains a password typo vault, and we prove that the construction above provides typo privacy (see Appendix H for the details and security proofs.) We also showed that the TypTop system does not provide "typo privacy"(see Section G.0.1). Intuitively, in the typo privacy game, the attacker gets to specify an initial password $pw_u$ for the user. The goal of the attacker is to predict a random bit $b$ selected by the challenger. The adversary may repeatedly either (1) submit a login query $pw'$ to the authentication server, or (2) submit a pair $(pw'_0, pw'_1)$ of guesses with $P_{typo}(pwd_u, pw'_0) = 0 = P_{typo}(pwd_u, pw'_1)$ to the challenger who will then forward the guess $pw'_b$ to the authentication server. The adversary is allowed to observed the state $\sigma_i$ of the authentication server immediately before $(\sigma_{i-1})$ and immediately after $(\sigma_i)$ each query $i$. Intuitively, if

the conditional encryption scheme is secure then the attacker should not be able to predict the secret bit $b$ since the only update after a type (2) query is to store the new ciphertext $c_b = \mathsf{CEnc}_{pk_u}(c_u, pw'_b, pw'_b)$. Since $P_{typo}(pwd_u, pw'_0) = 0 = P_{typo}(pwd_u, pw'_1)$ both $c_0 = \mathsf{CEnc}_{pk_u}(c_u, pw'_0, pw'_0)$ and $c_1 = \mathsf{CEnc}_{pk_u}(c_u, pw'_1, pw'_1)$ are indistinguishable from a random ciphertext $\mathtt{Sim}(pk)$ generated without knowledge of $pwd_u, pw'_1$ or $pw'_0$.

# 5 Implementation and Empirical analysis

In this section, we discuss our implementations of Conditional Encryption for the predicates $P_{=}, P_{\mathsf{CAPSLOCK}}, P_{\ell,\mathtt{Ham},\mathtt{Pad}}$ and $P_{\ell=1,\mathtt{ED}}$ as well as $P_{typo} = P_{\mathsf{CAPSLOCK}} \vee P_{\ell=2,\mathtt{Ham},\mathtt{Pad}} \vee P_{\ell=1,\mathtt{ED}}$. We also implement a modified version TypTop Personalized Typo Correction service to instantiate the Typo Vault using our conditional encryption scheme for the predicate $P_{typo}$. We empirically evaluate the performance of each implementation e.g., running time, ciphertext size etc.

## 5.1 Conditional Encryption

### 5.1.1 Implementation

We implemented our conditional encryption schemes in C++. The implementation is available on Github [AB24a] and Zenodo [AB24b]. Our implementation includes conditional encryption schemes for the following predicates: CAPSLOCK, Edit Distance One, Hamming Distance at most one, Hamming Distance at most two, as well as the OR of these predicates. We also implemented conditional encryption for a general Hamming Distance predicate for arbitrary distance thresholds $t = \{1, 2, 3, 4\}$. We defined our message space to be $\Sigma^{\leq n}$ where $\Sigma$ 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 1024-bit modulus (80-bit security), 2048-bit modulus (112-bit security) and 3072-bit modulus (128-bit security) [PDB+15]. We remark that for our applications to password typo vaults 80-bit security should be sufficient as it would *almost certainly* be easier for an offline attacker to brute-force the user's password and then extract the Pallier secret key directly than to factor a 1024-bit modulus $N$ e.g., see [Bon12, BL23, MUS+16]. **Construction** 18 and $\Pi_{1,\mathtt{ED}}$ (our edit-distance construction) requires that $|\Sigma|^{n+1} \leq \min\{p, q\}$. Thus, when our message length is $n = 64$ characters (resp. $n = 128$ characters) we must use a 2048-bit (resp. 3072-bit) modulus $N$ to ensure that $\left|\{0,1\}^8\right|^n = 2^{8(n+1)} < \min\{p, q\}$ since $\min\{p, q\} < \sqrt{N}$.

Like TypTop [CWP+17], our implementations of conditional encryption use CryptoPP [Dai16] for Authenticated Encryption and Shamir Secret Sharing. For authenticated encryption, we use AES-GCM with 128-bit keys and we use Shamir Secret Sharing over a field of size $2^{128}$ to generate shares of the secret symmetric key. Our code is available on Zenodo [AB24b][5].

### 5.1.2 Optimized Implementation of the Hamming Distance Predicate

We implemented several versions of our conditional encryption scheme for the Hamming Distance Predicate $P_{\ell,\mathtt{Ham},\mathtt{Pad}}$ to optimize performance. The (unoptimized) implementation follows **Construction** 19 without any optimizations. As noted previously the worst-case running time to decrypt a conditionally encrypted ciphertext is roughly $\binom{n}{\ell}(t_{SSrec} + t_{AE})$ where $t_{SSrec}$ (resp. $t_{AE}$) denotes the running time for $\mathsf{SecretRecover}$ over a field of size $2^\lambda$ (resp. $\mathsf{Auth.Dec}$).

We can make a simple optimization to speed up the running time of $\mathsf{Dec}$. In particular, we modify $\mathsf{CEnc}$ to generate $n$ shares $[\![z]\!]_1, \ldots, [\![z]\!]_n \leftarrow \mathsf{ShareGen}(n, n-\ell, 0)$ of 0 over a smaller field of size $2^{32} \ll 2^\lambda$ *in addition to* the $n$ shares $[\![s]\!]_1, \ldots, [\![s]\!]_n$ of our secret key $K$. For each character $i$ where $m_1[i] = m_2[i]$ the ciphertext $\tilde{c}_i$ will allow us to extract both shares $[\![s]\!]_i$ and $[\![z]\!]_n$. Thus, for each subset $S \subseteq [n]$ of size $|S|$ we can first compute $x_S = \mathsf{SecretRecover}(\{(i, [\![z]\!]_i)\}_{i \in S})$ by running $\mathsf{SecretRecover}$ over our smaller field.

[5] https://zenodo.org/uploads/13744111

Only if $x_S = 0$ do we then proceed to compute $K_S = \mathsf{SecretRecover}(\{(i, [\![s]\!]_i)\}_{i \in S})$ by running $\mathsf{SecretRecover}$ over our larger field and then attempt to decrypt our authenticated encryption ciphertext using $K_S$. We will still have to run $\mathsf{SecretRecover}$ over our smaller field $\binom{n}{n-\ell}$ times. However, in expectation we will only need to run $\mathsf{SecretRecover}$ over the large field (resp. $\mathsf{Auth.Dec}$) *at most* $1 + \binom{n}{n-\ell}2^{-32}$ times. Our empirical analysis indicates that this optimization significantly speeds up the worst-case running time of our decryption algorithm — see Figure 1a. For example, when $n = 32$ and $\ell = 4$ the optimized version of $\mathsf{Dec}$ is more than six times faster than the unoptimized version of $\mathsf{Dec}$.

Our second optimization exploits the simple observation that most user passwords are somewhat short. The goal of finding the subset $S \subseteq [n]$ of $|S| = n - \ell$ correct shares is equivalent to finding the set $C = \{i \in [n] : \mathtt{Pad}(m_1)[i] \neq \mathtt{Pad}(m_2)[i]\} \subseteq [n]$ of corrupted shares. Suppose that we know $m_1, m_2 \in \Sigma^{\leq k}$ are both shorter passwords of length at most $k < n$ and that $\mathtt{Ham}(\mathtt{Pad}(m_1), \mathtt{Pad}(m_2)) \leq \ell$. In this case there would only be $\binom{k}{\ell} \ll \binom{n}{\ell}$ possible choices of $C$ to check. In the breached RockYou password dataset 99% (resp. 99.9 %) of passwords were shorter than 15 (resp. 30) characters. Thus, if we expect that most of the inputs are short we can optimize the decryption algorithm by iterating from $k = \ell$ to $n$, iterating over all $\binom{k-1}{\ell-1}$ subsets $C' \subseteq [k-1]$ of size $\ell - 1$, setting $C = C' \cup k$, $S = [n] \setminus C$ and then running $\mathsf{SecretRecover}$ with the shares in $S$. In our password typo application we will consider the Hamming Distance predicate with distance parameter $\ell = 2$. Examining the password typo dataset collected by Chatterjee et al. [CAA+16] we observed that in over 80% of the instances where the predicate $P_{\ell=2,\mathtt{Ham}}$ holds that the Hamming Distance was actually just 1. If there is only one invalid share then we are guaranteed to find the correct secret after just $n/2$ attempts by first running $\mathsf{SecretRecover}$ with the shares $S = [n] \setminus C$ for each $C \in \{\{2i - 1, 2i\} : 1 \leq i \leq n/2\}$.

This optimization significantly improved the conditional decryption algorithm when the padding size is larger like $n \in \{32, 64, 128\}$. As an example, if we consider Hamming distance with $\ell = 4$ and $n = 32$, we observe that the decryption algorithm takes 14.664 seconds for our unoptimized implementation, while the average running time (using random RockYou passwords) is reduced to just 205.69 *milliseconds* when both optimizations are applied.

### 5.1.3 Evaluation

We evaluated the performance of our implementation of conditional encryption on a Lenovo ThinkStation S30 with a 2.9 GHz 8-core Intel® Xeon® E5-26900x 16 CPU processor and 28 GB DDR4 RAM memory. Figure 1 and Table 1 shows the running time for $\mathsf{C.KeyGen}$, $\mathsf{Enc}$ ,$\mathsf{CEnc}$, $\mathsf{CDec}$ (we slightly abuse notation and use $\mathsf{CDec}$ to refer to the decryption algorithm $\mathsf{Dec}$ when the input is a conditional ciphertext) as well as the ciphertext size for the aforementioned predicates. The primary difference between Table 1 and Figure 1 are as follows (1) Figure 1 plots the worst-case running time for $\mathsf{CDec}$ (when the relevant predicates do not hold) while the performance analysis in Table 1 is based on empirical user typos i.e., we evaluate the running time $\mathsf{CDec}$ by selecting random password/typo pairs from the password typo dataset of Chatterjee et al. [CAA+16] subject to the constraint that the relevant predicate holds. (2) Table 1 focuses exclusively on conditional encryption schemes for messages of length at most $n = 32$ i.e., the parameter that we use for TypTop.

For the Hamming distance, we consider four different thresholds at most one, at most two, at most three and at most four. In Figure 1a and Figure 1b we focus on the worst case running time for $\mathsf{CDec}$ when the predicate does not hold and we have to iterate over all $\binom{n}{\ell}$ possible subsets for secret recovery. Figure 1a plots the running time of $\mathsf{CDec}$ as the input length $n$ varies for different Hamming Distance thresholds $\ell \in \{1, 2, 3, 4\}$. Figure 1b plots how the running time of $\mathsf{CDec}$ is impacted by the Hamming Distance threshold $\ell$. The figure includes separate plots for messages of length $n \in \{8, 16, 32, 64, 128\}$. In figures Figure 1a and Figure 1b the blue (resp. red) curves highlight the running time of our optimized (resp. non-optimized) implementation. Figure 1c plots the running time of the encryption and conditional encryption algorithms $\mathsf{Enc}$ and $\mathsf{CEnc}$ and Figure 1h plots the size of a regular and conditional ciphertext for the Hamming Distance predicate as the message length varies. As expected we note that the ciphertext size is independent of the threshold $\ell$ and that the size of a conditional ciphertext is approximately equal to the size of a regular ciphertext.

We did similar for the CAPSLOCK predicate and considered the evaluation time over different message

Table 1: Conditional Encryption: Computation Time and Ciphertext Size ($n = 32$, 80-bit security)

| | Enc | | CEnc | | CDec |
|---|---|---|---|---|---|
| **Predicate:** | Time (ms) | $\|c\|$ | Time (ms) | $\|c\|$ | Time (ms) |
| EdDist One | 108.68 | 8.27 | 406.842 | 16.29 | 104.31 |
| HamDist ($\ell = 1, n = 32$) | 85.582 | 8.01 | 412.424 | 8.04 | 85.644 |
| HamDist ($\ell = 1, n = 32$) OPT | 92.384 | 8.01 | 445.714 | 8.04 | 263.626 |
| HamDist ($\ell = 2, n = 32$) | 93.88 | 8.01 | 445.8 | 8.04 | 347.953 |
| HamDist ($\ell = 2, n = 32$) OPT | 98.0633 | 8.01 | 475.58 | 8.04 | 264.273 |
| HamDist ($\ell = 3, n = 32$) | 90.1867 | 8.01 | 433.63 | 8.04 | 2268.54 |
| HamDist ($\ell = 3, n = 32$) OPT | 105.98 | 8.01 | 498.75 | 8.04 | 254.61 |
| HamDist ($\ell = 4, n = 32$) | 97.52 | 8.01 | 461.79 | 8.04 | 14664.8 |
| HamDist ($\ell = 4, n = 32$) OPT | 98.77 | 8.01 | 466.457 | 8.04 | 205.69 |
| CAPSLOCK on | 3.0025 | 0.27 | 13.26 | 0.29 | 1.01 |
| OR* | 201.15 | 16.54 | 900.945 | 24.64 | 360 |

$|c|$ = Ciphertext size (KB)
* OR = EditDistOne or HamDistTwo or CAPSLOCKon
** $P_i$ is the predicate and we define our CondCrypto over this predicate for $i = \{1, 2, 3, 4\}$, which implies 4 different predicates.
*** For hamming distance (HamDist), $\ell$ represents the threshold value and $n = 32$ is the padding size. Also, OPT means using optimized decryption algorithm.

lengths $n = \{8, 16, 32, 64, 128\}$ and the average time of each algorithm is presented in Figure 1e. The running time for each algorithm Enc, CEnc and Dec is independent of the message length until we have to increase the size of our Pallier Public key to satisfy the requirement that $|\Sigma|^{n+1} \leq \min\{p, q\}$. This explains the jumps at input length 64 and 128.

Figure 1d plots the running time of Enc, CEnc and CDec for our edit distance one predicate under different padding lengths $n = \{8, 16, 32, 64, 128\}$. Similarly, Figure 1f plots the running time of of Enc, CEnc and CDec for the OR predicate $P_{typo}$. For CDec we report the worst-case running time to decrypt a conditionally encrypted ciphertext i.e., when the predicate does not hold. When eveluating decryption time for the OR predicate $P_{typo}$ we use our optimized implementation of conditional decryption for the hamming distance predicate.

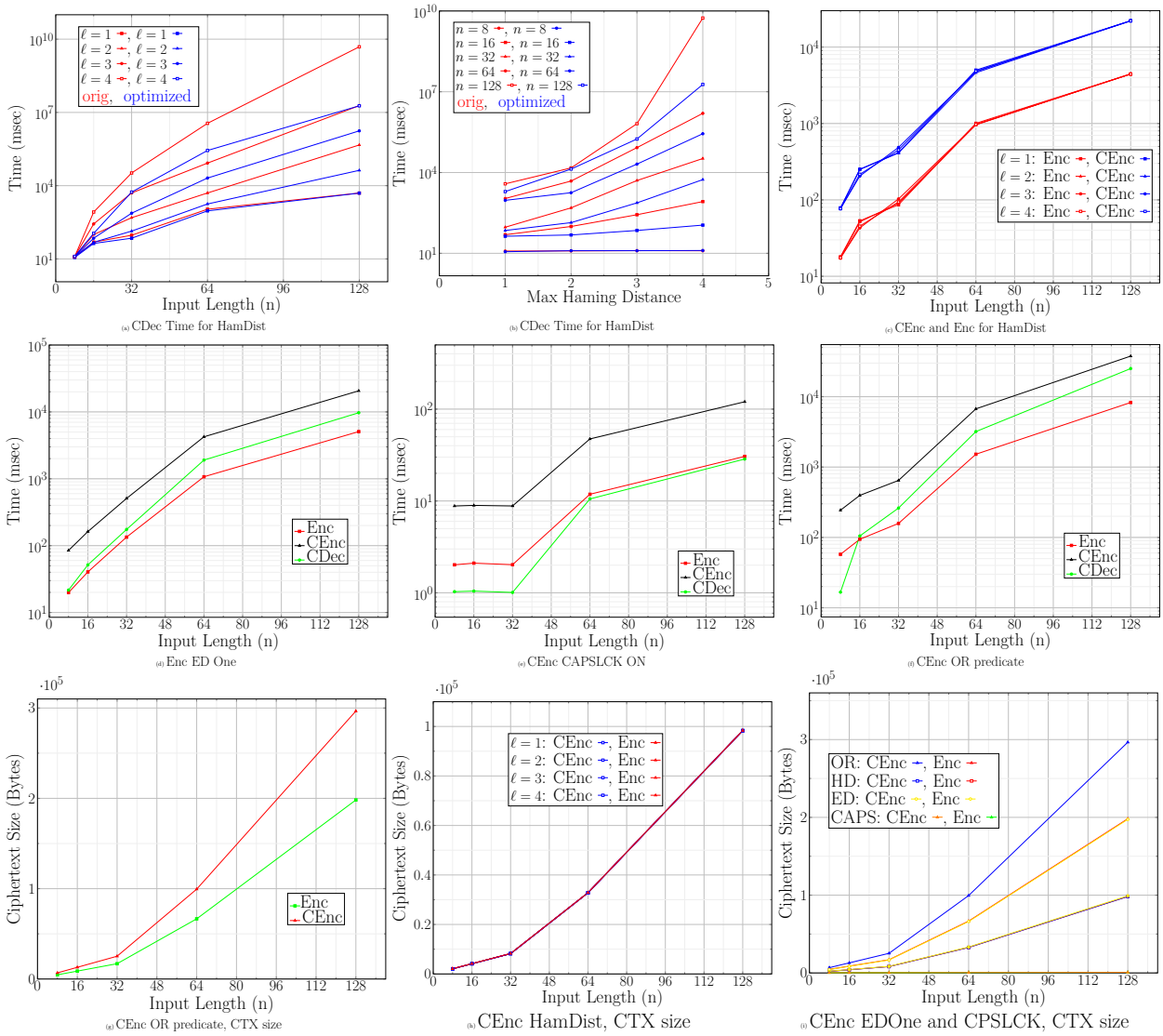
Figure 1i plots the size of a regular and conditional ciphertexts as the message length increases for each predicate: CAPSLOCK (CAPS), Edit Distance One (ED), Hamming Distance Two (HD) and the OR of the above. Some plots are difficult to see because they are identical to other lines. For example, we first note that the size of a regular ciphertext for the OR predicate is identical to the size of a ED ciphertext. Similarly, the size of a conditional ciphertext is approximately equal for the Hamming Distance (HD) and Edit Distance (ED) predicates. The plots at the bottom of Figure 1i are for CAPS as a regular/conditional encryption for this predicate consists of a single Pallier ciphertext.

#### 5.1.4 Discussion

Our empirical analysis demonstrates the practicality of our constructions especially for password typos. For example, when $n = 32$ the worst-case time to decrypt a conditional ciphertext for the password typo predicate $P_{typo}$ (OR) is less than 250 (ms). While the overhead is higher than traditional encryption schemes, it is important to note that, for our TypTop application, the algorithms CEnc and Dec can be evaluated offline and will not delay user authentication.

### 5.2 TypTop with Typo Privacy

We also implemented a modified version of TypTop system for personalized typo correction [CWP+17] as outlined in Section 4. We consider two primary modifications to the regular TypTop system. First, we replace the Key Derivation Function (KDF) with Argon2id [BDK16] a Memory-Hard Key Derivation Function. This modification was already suggested by the designers of TypTop. Second, we replace the




(a) CDec Time for HamDist (b) CDec Time for HamDist (c) CEnc and Enc for HamDist

(d) Enc ED One (e) CEnc CAPSLCK ON (f) CEnc OR predicate

(g) CEnc OR predicate, CTX size (h) CEnc HamDist, CTX size (i) CEnc EDOne and CPSLCK, CTX size

Figure 1: Performance Evaluation for our proposed Conditional Encryption schemes under different, predicates, lengths and distances

Table 2: TypTop: Computation and Storage Overhead with/without Conditional Encryption. Here we considered 80-bit level of security when $n = 32$.

| | | Correct login | | Incorrect login | | |
|---|---|---|---|---|---|---|
| | Init | Auth Delay | Total Running Time | Auth Delay | Total Running Time | Storage (KB) |
| Typtop [CWP+17] | 171.95 (ms) | 26.41 (ms) | 53.384 (ms) | 156.556 (ms) | 158.71 (ms) | 1 |
| CondTyptop | 6.771 (s) | 25.7 (ms) | 11.203 (s) | 160.3 (ms) | 0.617 (s) | 246 |
| CondTyptop (Optimized) | 7.13 (s) | 24.12 (ms) | 8.690 (s) | 160.33 (ms) | 0.629 (s) | 246 |
| Typtop (mhf) [CWP+17] | 5.738 (s) | 0.943 (s) | 0.784 (s) | 5.644 (s) | 5.856(s) | 1 |
| CondTyptop(mhf) | 19.672 (s) | 0.933 (s) | 16.558 (s) | 5.446 (s) | 6.162 (s) | 246 |
| CondTyptop(mhf/opt) | 14.456 | 0.729 (s) | 10.995 | 4.376(s) | 5.87 (s) | 246 |

* CondTyptop is the modified typtop scheme when conditional encryption is used.

public key encryption scheme with a conditional encryption scheme for the OR predicate. For the purpose of empirical evaluation we implement TypTop with our optimized implementation of conditional encryption and with our unoptimized implementation refering to these as modifications (2A) and (2B) respectively.

Our code is available at [AB24a]. In our analysis we analyze six versions of the TypoTop system: the original system (no modifications), modification (1) only, modification (2A) only, modification (2B) only, modifications (1)+(2A) and finally modifications (1)+(2B). The recommended version is TypTop with modifications (1)+(2A) i.e., using the Memory-Hard Key Derivation Function and the optimized implementation of our conditional encryption scheme for the OR predicate.

In our modified implementation of TypoTop, we assume that the length of each user passwords is at most 32 characters. This is a valid assumption as 99.9 % of the passwords has length of lower than 32 characters. To support this assumption, as an evidence we extracted this stats from the leaked passwords of LinkedIn and RockYou. More specifically, 99% of passwords from LinkedIn Frequency Corpus[6] as well as passwords from RockYou[7] have length at most 15. If desired one could easily adjust the TypTop system to support longer passwords. However, the scheme would either become less efficient or we would need to leak a single bit of information about the user password i.e., indicating whether or not the length of the password is more than 32 characters.

In our empirical analysis we register a user password of length $\leq 32$ and then generate a sequence of 1000 correct and 1000 incorrect login requests. Incorrect login requests are randomly selected as (1) CAPSLOCK error, (2) Hamming Distance $\leq 2$, (3) Edit Distance $\leq 1$ or (4) completely different. We analyze the running time for Initialization, Correct Login attempt and for Incorrect login attempts — see Table 2. In our analysis we distinguish between authentication delay and total running time. For example, if authentication fails with an incorrect password $pw'_u$ then we will want to run our conditional encryption algorithm $\mathsf{CEnc}_{pk_u}(c_u, pw'_u, pw'_u)$ so that, if $pw'_u$ is close enough to the real password, we can recover it at a later point in time. However, we can immediately inform the user of the outcome of authentication before performing this somewhat expensive computation. Similarly, if a user logs in with the correct password $pw_u$ then we can recover the symmetric key $K_u$ and decrypt $sk_u$. At this point we will want to decrypt all of the conditional ciphertexts in our vault, but again we can immediately inform the user that the authentication attempt was successful before decrypting these conditional ciphertexts.

### 5.2.1 Discussion

We can use conditional encryption to strengthen the security guarantees of TypTop without increasing authentication delay for users. The usage of conditional encryption does increase offline computation and

[6]Link to the data set: https://figshare.com/articles/dataset/linkedin_files_zip/7350287

[7]Link to the RockYou with count dataset: https://github.com/danielmiessler/SecLists/tree/master/Passwords/Leaked-Databases

storage requirements, but the overhead is still manageable. The reason why the offline computation is higher after a correct login attempt is because this allows us to recover the secret decryption key and then decrypt all of the conditional ciphertexts in our waitlist so that we can consider adding them to our cache of acceptable typos. It is worth noting that the TypTop system maintains the invariant that there are always 10 (conditional) ciphertexts in the waitlist. The invariant, which is maintained by seeding the waitlist with dummy ciphertexts, ensures that an attacker cannot infer the number of incorrect login attempts. If we don't maintain this invariant then an attacker who breaches the authentication server would learn the total number of incorrect login attempts that were submitted since the last correct login. Our modified implementation of TypTop maintains the same invariant. However, such leakage should arguably not be viewed as problematic since it does not reveal anything about incorrect login attempts or the user's password. In this case we could reduce offline computation (and storage) of TypTop by only placing conditional ciphertexts in the waitlist when there is an incorrect login attempt i.e., if there were no incorrect login attempts since the last correct login attempt then there would be no offline work to decrypt the conditional ciphertexts in the waitlist because the waitlist would be empty!


### Acknowledgements

This work was supported in part by the National Science Foundation under CAREER Award CNS 2047272. Any views expressed in this paper are those of the authors and do not necessarily represent the position of the National Science foundation. The authors wish to thank anonymous CCS reviewers for constructive feedback and for the suggestion to consider circuit private FHE.


## References


[AB17] Joël Alwen and Jeremiah Blocki. Towards practical attacks on argon2i and balloon hashing. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157. IEEE, 2017.

[AB24a] Mohammad Hassan Ameri and Jeremiah Blocki. Implementation of conditional encryption and typotop, 2024. April, 26.

[AB24b] Mohammad Hassan Ameri and Jeremiah Blocki. Implementation of conditional encryption and typotop, September 2024.

[ABP18] Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 99–130, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

[ADMS18] Mohammad Hassan Ameri, Mahshid Delavar, Javad Mohajeri, and Mahmoud Salmasizadeh. A key-policy attribute-based temporary keyword search scheme for secure cloud storage. *IEEE Transactions on Cloud Computing*, 8(3):660–671, 2018.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 500–518, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[AS15] Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 595–603, Portland, OR, USA, June 14–17, 2015. ACM Press.

[AYY22] Shweta Agrawal, Anshu Yadav, and Shota Yamada. Multi-input attribute based encryption and predicate encryption. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part I*, volume 13507 of *Lecture Notes in Computer Science*, pages 590–621, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

[BCFG17] Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 67–98, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.

[BDK16] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229, Santa Barbara, CA, USA, August 19–23, 2001. Springer, 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, Heidelberg, Germany.

[BGK08] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 2008: 15th Conference on Computer and Communications Security*, pages 417–426, Alexandria, Virginia, USA, October 27–31, 2008. ACM Press.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BHZ18] Jeremiah Blocki, Benjamin Harsha, and Samson Zhou. On the economics of offline password cracking. In *2018 IEEE Symposium on Security and Privacy*, pages 853–871, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[BL23] Jeremiah Blocki and Peiyuan Liu. Towards a rigorous statistical analysis of empirical password datasets. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 606–625. IEEE, IEEE, 2023.

[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.

[BRS13] Dan Boneh, Ananth Raghunathan, and Gil Segev. Function-private identity-based encryption: Hiding the function in functional encryption. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 461–478, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.

[BSW07] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy*, pages 321–334, Oakland, CA, USA, May 20–23, 2007. IEEE Computer Society Press.

[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.

[CC09] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In Ehab Al-Shaer, Somesh Jha, and Angelos D. Keromytis, editors, *ACM CCS 2009: 16th Conference on Computer and Communications Security*, pages 121–130, Chicago, Illinois, USA, November 9–13, 2009. ACM Press.

[CCW+21] Yunang Chen, Amrita Roy Chowdhury, Ruizhe Wang, Andrei Sabelfeld, Rahul Chatterjee, and Earlence Fernandes. Data privacy in trigger-action systems. In *2021 IEEE Symposium on Security and Privacy*, pages 501–518, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 515–534, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

[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, Heidelberg, Germany.

[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.

[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, Heidelberg, Germany.

[EM23] Johannes Ernst and Aikaterini Mitrokotsa. A framework for UC secure privacy preserving biometric authentication using efficient functional encryption. In Mehdi Tibouchi and Xiaofeng Wang, editors, *ACNS 23: 21st International Conference on Applied Cryptography and Network Security, Part II*, volume 13906 of *Lecture Notes in Computer Science*, pages 167–196, Kyoto, Japan, June 19–22, 2023. Springer, Heidelberg, Germany.

[Gen06] Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 445–464, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st Annual ACM Symposium on Theory of Computing*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

[GGHZ16a] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 480–511, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany.

[GGHZ16b] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In *Theory of Cryptography: 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II 13*, pages 480–511. Springer, 2016.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006: 13th Conference on Computer and Communications Security*, pages 89–98, Alexandria, Virginia, USA, October 30 – November 3, 2006. ACM Press. Available as Cryptology ePrint Archive Report 2006/309.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. *Journal of the ACM (JACM)*, 62(6):1–33, 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 503–523, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162, Istanbul, Turkey, April 13–17, 2008. Springer, Heidelberg, Germany.

[LW11] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 568–588. Springer, 2011.

[MUS+16] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016: 25th USENIX Security Symposium*, pages 175–191, Austin, TX, USA, August 10–12, 2016. USENIX Association.

[NAP+14] Muhammad Naveed, Shashank Agrawal, Manoj Prabhakaran, XiaoFeng Wang, Erman Ayday, Jean-Pierre Hubaux, and Carl A. Gunter. Controlled functional encryption. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014: 21st Conference on Computer and Communications Security*, pages 1280–1291, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[OPP14] Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. Maliciously circuit-private FHE. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 536–553, Santa Barbara, CA, USA, August 17–21, 2014. Springer, 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, 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, 2015.

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

[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, Heidelberg, Germany.

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

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010: 17th Conference on Computer and Communications Security*, pages 463–472, Chicago, Illinois, USA, October 4–8, 2010. ACM Press.

[SSW09] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473. Springer, Heidelberg, Germany, March 15–17, 2009.

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473, Aarhus, Denmark, May 22–26, 2005. Springer, Heidelberg, Germany.

[vGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43, French Riviera, May 30 – June 3, 2010. Springer, Heidelberg, Germany.

[VJH21] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. SoK: Fully homomorphic encryption compilers. In *2021 IEEE Symposium on Security and Privacy*, pages 1092–1108, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[WPC23] Yuyu Wang, Jiaxin Pan, and Yu Chen. Fine-grained secure attribute-based encryption. *Journal of Cryptology*, 36(4):33, October 2023.

# A 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 $\lambda$ and two uniformly at random sampled large $\mathsf{poly}(\lambda)$[8] bit prime numbers $p, q \leq 2^{\mathsf{poly}(\lambda)}$, and generates the secret-public key pair $(pk, sk)$ as follows. Then the algorithm 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 $\mu = \phi(N)^{-1} \mod N$.

[8] We should highlight that based on the security parameter we desire, there exists a polynomial function like $\mathsf{poly}$ which determined the bit length of the prime numbers.

- $c \leftarrow \mathtt{P.Enc}_{pk}(m; r)$: 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:

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

  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:

$$\begin{aligned} c &= (c_2)^{m_1} \mod N^2 = (g^{m_2} r_2^N)^{m_1} = g^{m_1 m_2}(r_2^{m_1})^N \mod N^2 \\ &= g^m r^N \mod N^2 \end{aligned} \tag{3}$$

  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 $m_1 \boxtimes c_2 = \mathtt{P.PlainToCtxMul}(m_1, c_2)$.

# B Details on Secret Sharing (SS)

Let $\mathbb{F}$ be a field of size $|\mathbb{F}| \geq n$. We define interpolation algorithm $\mathsf{InterPol}$ over $t$-degree polynomial $f : \mathbb{F} \to \mathbb{F}$ which takes as input $t$ tuples $(x_i, y_i = f(i))$ for $1 \leq i \leq t$, and outputs $f(0)$. So we have $f(0) = \mathsf{InterPol}((x_1, f(x_1)), \ldots, (x_t, f(x_t)))$. The $(n, t)$- secret sharing scheme for the secret message $s \in \mathbb{F}$ is defined based on two algorithms $\mathsf{ShareGen}$, $\mathsf{SecretRecover}$ which are described in what follows.

- $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \mathsf{ShareGen}(n, t, s)$. 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 $t$-degree polynomial $\phi : \mathbb{F} \to \mathbb{F}$ such that $s = \phi(0)$ and sets the shares as $\llbracket s \rrbracket_i = \phi(i)$, for all $1 \leq i \leq n$.

- $s \leftarrow \mathsf{SecretRecover}(\mathsf{shares} = (\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_{t'}))$. This algorithm takes the set of share $\mathsf{shares}$ of size $t'$ as input, and recover the message $s$ if there exists set of valid shares $\mathsf{ValidShare} \subset \mathsf{shares}$ s.t. $|\mathsf{ValidShare}| \geq t$. To recover $s$ the algorithm runs the interpolation algorithm $\hat{s} = \mathsf{InterPol}((1, \llbracket s \rrbracket'_1), \ldots, (t, \llbracket s \rrbracket'_t))$ such that $\llbracket s \rrbracket'_i, \in \mathsf{ValidShare}, \forall 1 \leq i \leq t$. Finally the algorithm outputs $s = \mathsf{ToInt}^{-1}(\hat{s})$.

**Definition 6** (Correctness)**.** *Given the $(t, n)$-secret sharing scheme $\Pi = (\mathsf{ShareGen}, \mathsf{SecretRecover})$, $\forall n \in \mathbb{N}$, $\forall s \in \mathcal{M}^n$ and $\forall S = \{i_1, \ldots, i_t\} \subseteq \{1, \ldots, n\}$ of size $t$, the correctness of $\Pi$ enforces that*

$$\Pr_{(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \mathsf{ShareGen}(1^\lambda, n, t, s)} [\mathsf{SecretRecover}(\llbracket s \rrbracket_{i_1}, \ldots, \llbracket s \rrbracket_{i_t}) = s] = 1 \tag{4}$$

**Definition 7** (SS Perfect Secrecy)**.** *The $(t,n)$-secret sharing scheme $\Pi = (\mathsf{ShareGen}, \mathsf{SecretRecover})$ is perfectly secrecy, if $\forall n \in \mathbb{N}$, $\forall s, s' \in \mathcal{M}^n$, $\forall S \subseteq \{1,\ldots,n\}$ s.t. $|S| < t$, and for all adversaries $\mathcal{A}$,*

$$\Pr_{(\llbracket s \rrbracket_1,\ldots,\llbracket s \rrbracket_n)\leftarrow \mathsf{ShareGen}(1^\lambda,n,t,s)}[\mathcal{A}(\llbracket s \rrbracket_i | i \in S) = 1]$$
$$= \Pr_{\mathcal{A}(\llbracket s' \rrbracket_1,\ldots,\llbracket s' \rrbracket_n)\leftarrow \mathsf{ShareGen}(1^\lambda,n,t,s')}[\mathcal{A}(\llbracket s' \rrbracket_i | i \in S) = 1] \quad (5)$$

We will use the Shamir Secret sharing scheme [Sha79]. Given any field $\mathbb{F}$ of size $|\mathbb{F}| \geq n$ the construction starts with $n+1$ distinct field elements $x_0, x_1, \ldots, x_n \in \mathbb{F}$. Given a secret $s \in \mathbb{F}$ we generate shares $\llbracket s \rrbracket_1 = (x_1, y_1), \ldots, \llbracket s \rrbracket_{t-1} = (x_{t-1}, y_{t-1})$ where $y_1, \ldots, y_{t-1} \in \mathbb{F}$ are random field elements. We then use polynomial interpolation to find a degree $t-1$ polynomial $p(x)$ such that $p(x_0) = s$ and $p(x_i) = y_i$ for $i \leq t-1$. Finally, for share $i > t-1$ we define $\llbracket s \rrbracket_i = (x_i, p(x_i))$.

One crucial property of Shamir Secret sharing that we rely on is that any subset of $t-1$ shares is uniformly random over $\mathbb{F}^{t-1}$. In particular, for all secrets $s \in \mathbb{F}$, all subsets $S = \{i_1, \ldots, i_{t-1}\} \subseteq [n]$ of size $t-1$ the shares $\llbracket s \rrbracket_{i_1}, \ldots, \llbracket s \rrbracket_{i_{t-1}}$ can be viewed as uniformly random independent elements in $\mathbb{F}$ unrelated to the secret $s$ i.e., for all $y \in \mathbb{F}^{t-1}$ we have

$$\Pr\left[\left(\llbracket s \rrbracket_{i_1}, \ldots, \llbracket s \rrbracket_{i_{t-1}}\right) = y\right] = |\mathbb{F}|^{-t+1}$$

where the randomness is taken over $(\llbracket s \rrbracket_1, \ldots, \llbracket s \rrbracket_n) \leftarrow \mathtt{ShareGen}(n,t,s)$.

# C Real or Random Security

Similar to the traditional public key encryption schemes, we require that the encryption scheme is secure against any computationally bounded adversary who does not have the secret key $sk$. In this section, we formally define Real-or-Random security based on the a security exeperiment/game called $\mathtt{CE}^{ROR}_{\mathcal{A}}(1^\lambda)$.

## C.1 RoR Experiment for Conditional Encryption

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 $\mathtt{CE}^{ROR}_{\mathcal{A}}(1^\lambda)$: (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 $\mathtt{CE}^{ROR}_{\mathcal{A}}(1^\lambda) = 1$ if and only if $b = b'$.

**Definition 8.** *(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* $\Pr\left[\mathtt{CE}^{ROR}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \epsilon(t(\lambda), q(\lambda), \lambda)$.

**Theorem 9.** *Assume that Pallier encryption is $(t(\lambda), q(\lambda), \epsilon(t(\lambda), q(\lambda), \lambda))$-real or random secure. Then **Construction 18** is $(t'(\lambda), q'(\lambda), \epsilon'(t'(\lambda), q'(\lambda), \lambda))$-real or random security with $t'(\lambda) = t(\lambda) - q'(\lambda)\,\mathrm{poly}(\lambda)$, $q'(\lambda) = q(\lambda)$ and $\epsilon'(t'(\lambda), q'(\lambda), \lambda) \leq 3\epsilon(t(\lambda), q(\lambda), \lambda)$.*

**Proof of Theorem 9:** (Sketch) Note that we can assume WLOG that the query to the $\mathsf{CEnc}$ oracle is of the form $((0, c_1), m_2, m_3)$ where $c_1 \in \mathbb{Z}^*_{N^2}$. Otherwise, if $\gcd(c_1, N) \neq 1$ or if the query has the form $((1, c_1), m_2, m_3))$ then the response will simply be $\perp$ regardless of the secret bit $b$. Let $c_1 \in \mathbb{Z}_{N^2}$ be given (We do *not* assume that $c_1 \in \mathbb{Z}^*_{N^2}$) and consider the query $((0, c_1), m_2, m_3)$ to $\mathsf{CEnc}$.

We note that if $b = 0$ the query returns $(1, c) = \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ a random Pallier ciphertext of the message $R(m_1 - m_2) + m_3 \mod N$ if $b = 1$ the query returns $(1, c) = \mathsf{CEnc}_{pk}(c_1, r_2, r_3)$ a random Pallier ciphtext for the the different message $R(m_1 - r_2) + r_3 \mod N$. We define four hybrids: H0.A, H0.B, H1.A and H1.B. In Hybrid H0.A (resp. H1.A) we play the ROR security game with bit $b = 0$ (resp. $b = 1$). In Hybrid H0.B (resp. H1.B) we continue to use the bit $b = 0$ (resp. $b = 1$) for the encryption oracle $\mathsf{Enc}$, but we replace the conditional encryption oracle $\mathsf{CEnc}$ oracle with an oracle that simply returns a random pallier ciphertext i.e., on input $(c, m_2, m_3)$ the oracle simply outputs $(1, (1+N)^{r_1} r_2^N \mod N^2)$ for $r_1 \in \mathbb{Z}_N$ and $r_2 \in \mathbb{Z}_N^*$. Intuitively, by ROR security for Pallier the attacker cannot distinguish between Hybrid H0.B and H1.B except with advantage $\epsilon(t, q, \lambda)$ i.e., is not the regular ROR security game $b = 0$ or $b = 1$ adding a useless extra oracle which returns random Paillier ciphertexts — this oracle could easily be simulated. By similar reasoning the attacker cannot distinguish between hybrid H0.A and H0.B (or H0.A and H0.B) except with advantage $\epsilon(t, q, \lambda)$. Intuitively, if the attacker $\mathcal{A}$ distinguishes hybrids H0.A and H0.B we can construct an attacker $\mathcal{B}$ for the Paillier ROR security game i.e., $\mathcal{B}$ simply runs $\mathcal{A}$ and anytime $\mathcal{A}$ submits a query $(c_1, m_2, m_3)$ we submit the query $m_3$ to the Paillier Encryption oracle to obtain $c_3$ and then return $c = c_3 c_1^R (1+N)^{-m_2 R} \mod N^2$. If $c_3$ was a random Paillier encryption of $m_3$ then $c$ is distributed exactly like $\mathsf{CEnc}(c_1, m_2, m_3)$ in H0.A (resp. H1.A). On the other hand if $c_3$ is a random Paillier ciphertext of a random message then $c$ is uniformly random ciphertext as in H0.B (resp. H1.B). The running time for the attacker $\mathcal{B}$ is at most $t(\lambda) = t'(\lambda) + q(\lambda)\,\mathrm{poly}(\lambda)$ where the term $q(\lambda)\,\mathrm{poly}(\lambda)$. Thus, attacker $\mathcal{B}$ distinguishes the hybrids H0.B (resp. H1.B) with probability at most $\epsilon(t(\lambda), q(\lambda), \lambda)$. Thus, any attacker running in time $t'(\lambda)$ distinguishes H0.A ($b = 0$) from the final hybrid H1.A ($b = 1$) with advantage at most $3\epsilon(t(\lambda), q(\lambda), \lambda)$. □

**Theorem 10.** *Assume that Pallier encryption is $(t(\lambda), q(\lambda), \epsilon(t(\lambda), q(\lambda), \lambda))$-real or random secure and that $\Pi_{AE}$ is $(t_{AE}(\lambda), q_{AE}(\lambda), \epsilon_{AE}(t_{AE}(\lambda), q_{AE}(\lambda), \lambda))$ real or random secure. Then **Construction 19** is $\left(t'(\lambda), q(\lambda)', \epsilon'(t'(\lambda), q(\lambda), \lambda)\right)$-real or random secure with $t'(\lambda) = \min\{t(\lambda), t_{AE}(\lambda)\} - o(n)$, $q'(\lambda) = \min\left\{q(\lambda)/(4n), \frac{q_{AE}(\lambda)}{2}\right\}$ and $\epsilon'(t'(\lambda), q'(\lambda), \lambda) = \epsilon(t(\lambda), q(\lambda), \lambda) + \epsilon_{AE}(t_{AE}(\lambda), q_{AE}(\lambda)', \lambda)$*

**Proof of Theorem 10:** (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 $(1, \tilde{c}_1, \ldots, \tilde{c}_n, c_{AE}) = \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ where each $\tilde{c}_i = c^{R_i} \left(g^{-m_2[i]} r_{2,i}^N\right)^{R_i} g^{m_3[i]} r_{3,i}^N \mod N^2$ for uniformly random $R_i \in \mathbb{Z}_N$ and $r_{2,i}, r_{3,i} \in \mathbb{Z}_N^*$. By ROR security for Pallier we can apply a Hybrid argument can swap $g^{-m_2[i]} r_{2,i}^N$ with $g^{r_{1,i}} r_{2,i}^N$ with $r_{1,i} \in \mathbb{Z}_N$ (resp. $g^{r_{4,i}} r_{2,i}^N$ ) with $r_{1,i} \in \mathbb{Z}_N$ (resp. $r_{4,i} \in \mathbb{Z}_N$) uniformly random. After $2n$ hybrids we have replaced each $\tilde{c}_i$ with a ciphertext of the form $c^{R_i} \left(g^{r_{i,1}} r_{2,i}^N\right)_i^R g^{r_{4,i}} r_{3,i}^N \mod N^2$. 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 the $b = 1$ case $\mathsf{CEnc}_{pk}(c_1, m_2', m_3')$ for random messages $m_2'$ and $m_3'$. Thus, we adjust $q'(\lambda) = q(\lambda)/(4n)$ since we invoke ROR security for Pallier during $4n \times q'(n)$ hybrids. Similarly, we invoke ROR security for authnticated encryption in $2 \times q(\lambda)'$ hybrids. □

**Theorem 11.** *Suppose that we are given $k$ separate conditional encryption schemes $\Pi_1, \ldots, \Pi_k$ corresponding to predicates $P_1, \ldots, P_k$ and that each $\Pi_i$ provides $(t_i(\lambda), q_i(\lambda), \epsilon_i(t_i(\lambda), q_i(\lambda), \lambda))$ real or random security. Then the construction $\Pi_{or}$ provides $(t(\lambda), q(\lambda), \epsilon(t(\lambda), q(\lambda), \lambda))$ real or random security with $\epsilon(\lambda) = \sum_{i=1}^k \epsilon_i(t_i(\lambda), q_i(\lambda), \lambda))$, $q(\lambda) = \min\{\frac{q_1(\lambda)}{2k}, \cdots, \frac{q_k(\lambda)}{2k}\}$ and $t(\lambda) = \min\{t_1(\lambda), \cdots, t_k(\lambda)\} - O(k \cdot \max(t_{\mathsf{CEnc}_1}, \cdots, t_{\mathsf{CEnc}_k}))$.*

**Proof of Theorem 11:** (Sketch) Intuitively, for each conditional encryption query $(c_1 = (c_{1,\Pi_1}, \cdots, c_{k,\Pi_k}), m_2, m_3)$, we need to query $\tilde{c}_i = \Pi_i.\mathsf{CEnc}(c_{1,\Pi_i}, m_2, m_3)$ if $b = 1$. So we have $k$ hybrids and in each hybrid we use the real or random security of one predicate. So the adversary's advantage is bounded by $\sum_{i=1}^k \epsilon_i(t_i(\lambda), q_i(\lambda), \lambda)$. For each query, the simulator need to compute $\tilde{c}_i$ which takes $t_{\mathsf{CEnc}_i}$. So considering all $k$ hybrids, the loss in time is bounded by $O(k \cdot \max\{t_{\mathsf{CEnc}_1}, \cdots, t_{\mathsf{CEnc}_k}\})$ after

these $k$ hybrids. We note that the number of queries for the resulting adversary is bounded by $\min\{q_1(\lambda), \cdots, q_k(\lambda)\}$. We can then reverse the hybrids to move to transition from $\Pi_{or}.\mathsf{CEnc}\ (c_1, m_2, m_3)$ (when $b = 1$) all the way to $b = 0$ case $\Pi_{or}.\mathsf{CEnc}(c_1, m_2', m_3')$ for random messages $m_2'$ and $m_3'$. □

# D General construction of Conditional Encryption from Circuit-Private FHE

In this section we show how one can construct conditional encryption for an arbitrary binary predicate $P$ using Circuit-Private FHE assuming that the predicate $P$ can be implemented as a polynomial sized circuit. The basic idea is to define a circuit $C_{P,m_2,m_3}(m_1)$ which outputs $m_3$ if $P(m_1, m_2) = 1$ and outputs 0 if $P(m_1, m_2) = 0$. Now the the conditional encryption algorithm $\mathsf{CEnc}_{pk}(c, m_2, m_3)$ simply runs $\mathsf{FHE.Eval}_{pk}(C_{P,m_2,m_3}, c)$. If the input ciphertext $c$ corresponds to a message $m$ such that $P(m, m_2) = 1$ then $\mathsf{FHE.Eval}_{pk}(C_{P,m_2,m_3}, c)$ will output a ciphertext $c'$ which decrypt to $C_P(m, m_2, m_3) = m_3$. Otherwise, if $P(m, m_2) = 0$ we get a ciphertext $c'$ which decrypts to 0. This construction is formalized in **Construction** 12.

We show that **Construction** 12 provides conditional encryption secrecy as long as the FHE construction satisfies the notion of *circuit privacy* [OPP14] — see Appendix D.2 for the formal definition. Before defining *circuit privacy* for FHE we first show that circuit privacy is necessary to prove security of our construction. In particular, if FHE exists then there exists a (non-circuit private) FHE scheme for which **Construction** 12 is not secure — see Appendix D.1. After defining circuit privacy in Appendix D.2 we use circuit privacy to prove that **Construction** 12 satisfies conditional encryption secrecy Theorem 13 in Appendix D.3.

More precisely, let $\mathsf{FHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ be an FHE scheme, then we obtain a conditional encryption scheme $\Pi_{\mathsf{CEnc}}$ as described in **Construction** 12 and Figure 2.

## D.1 Circuit Privacy is Necessary

We demonstrate that **Construction** 12 can be insecure if the underlying FHE scheme does not satisfy circuit privacy. In particular, there exists secure FHE schemes for which the proposed construction is blatantly insecure as a conditional encryption scheme. Security definitions for FHE assume that the attacker does not have the secret decryption key, and there is no requirement that the output $ct = \mathsf{Eval}(C, c_1 \ldots, c_n)$ hides information about the underlying message bits $m_1, \ldots, m_n$ when the secret key is known. Given a FHE scheme $\Pi_{\mathsf{FHE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ we can define a new FHE scheme $\Pi'_{\mathsf{FHE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}', \mathsf{Dec}')$ where $\mathsf{Eval}'(C, c_1, \ldots, c_n) = (\mathsf{Eval}(C, c_1, \ldots, c_n), c_1)$ i.e., $\mathsf{Eval}'$ simply appends the first input ciphertext $c_1$ to the final output. Correctness still holds as the decryption algorithm $\mathsf{Dec}'$ can simply ignore $c_1$ and then run the original decryption algorithm $\mathsf{Dec}$. Note that as long as the attacker does not have the secret key that the output of $\mathsf{Eval}'$ does not leak any information about the encrypted message $m_1$ corresponding to $c_1$. Thus, the updated FHE scheme $\Pi'_{\mathsf{FHE}}$ still satisfies the traditional notion of semantic security. However, the message $m_1$ can be directly extracted from the evaluation ciphertext $ct' = \mathsf{Eval}'(C, c_1, \ldots, c_n)$ by any party who has the secret decryption key. Instantiated with $\mathsf{Eval}'$ the proposed conditional encryption scheme would output $\mathsf{CEnc}_{pk}(c_1, m_2, m_3) = \mathsf{FHE.Eval}'_{pk}(C_{P,m_2,m_3}, c_1) = (\mathsf{FHE.Eval}_{pk}(C_{P,m_2,m_3}, c_1), c_1) =$. Supposing that $P(m_1, m_2) = 0$ conditional encryption secrecy requires that the conditional ciphertext $ct' = \mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ leaks no information about any of the messages $m_1, m_2$ and $m_3$ except that $P(m_1, m_2) = 0$. Yet, the ciphertext $ct'$ still contains an encryption of the secret message $m_1$. In our setting the attacker is given the decryption key allowing the attacker to directly extract the secret message $m_1$ from $ct'$. Thus, **Construction** 12 blatantly violates conditional encryption secrecy when instantiated with $\Pi'_{\mathsf{FHE}}$. If we want to show that **Construction** 12 is secure we will need to rely on additional security properties for FHE (circuit privacy) which rules out FHE constructions like $\Pi'_{\mathsf{FHE}}$.

**Construction 12.**

Key Generation Algorithm:

- $(pk, sk) \leftarrow \Pi_{\mathsf{CEnc}}.\mathsf{KeyGen}(1^\lambda) = \mathsf{FHE}.\mathsf{Setup}(1^\lambda)$

Regular Encryption:

- $(0, c) \leftarrow \Pi_{\mathsf{CEnc}}.\mathsf{Enc}_{pk}(m_1) = \mathsf{FHE}.\mathsf{Enc}_{pk}(m_1)$

Conditional Encryption algorithm:

- $(1, c') \leftarrow \Pi_{\mathsf{CEnc}}.\mathsf{CEnc}_{pk}(c, m_2, m_3)$:
  (1) Define the circuit $C_{P,m_2,m_3}(m_1)$ which outputs $m_3$ if $P(m_1, m_2) = 1$ and outputs $Z$ if $P(m_1, m_2) = 0$. (Note: $Z$ is an arbitrary fixed constant which is publicly known. If we want the construction to be error detecting then $Z$ needs be outside of the message space for our conditional encryption scheme, but inside the FHE message space.
  (2) Run $c' \leftarrow \mathsf{FHE}.\mathsf{Eval}_{pk}(C_{P,m_2,m_3}, c)$
  (3) Return $(1, c')$

Decryption algorithm:

- $m := \Pi_{\mathsf{CEnc}}.\mathsf{Dec}_{sk}(b, c)$
  (1) Compute $m = \mathsf{FHE}.\mathsf{Dec}_{sk}(c)$
  (2) If $m$ is outside the message space return $\perp$; otherwise return $m$

Figure 2: Proposed generic construction of our generic construction of Conditional Encryption from FHE

## D.2 Circuit Privacy Definition

To prove security of **Construction** 12 we rely on the notion of circuit privacy. Gentry proposed a formal definition of (statistical) circuit private FHE in his thesis [Gen09a] and suggested how to achieve circuit privacy. Ostrovsky et al. [OPP14] introduced the notion maliciously circuit private FHE which requires that privacy holds even when the public key and/or input ciphertexts are generated maliciously. We base our definition of circuit privacy on Gentry's definition of circuit privacy as malicious security is not necessary to show conditional encryption secrecy.

Intuitively, a FHE scheme is circuit private if the ciphertexts $c = \mathsf{FHE.Eval}_{pk}(C, c_1, \ldots, c_n)$ reveals nothing about the circuit $C$ or the encrypted messages $m_1, \ldots, m_n$ corresponding to the input ciphertexts $c_1, \ldots, c_n$ other than the output $C(m_1, \ldots, m_n)$ of the circuit even if the adversary has the secret key or is computationally unbounded (statistical security). This intuition is formalized by the notion of a simulator who takes as input the public key $pk$ and the circuit output $C(m_1, \ldots, m_n)$ and generates a ciphertext that is statistically indistinguishable from $\mathsf{FHE.Eval}_{pk}(C, c_1, \ldots, c_n)$ — in [Gen09a, Def 2.1.6] the simulator is defined to be $\mathtt{Sim}(pk, C(m_1, \ldots, m_n)) = \mathsf{FHE.Enc}_{pk}(m_1, \ldots, m_n)$. We slightly rephrase the definition [Gen09a] to obtain a concrete security definition instead of an asymptotic definition.

**Definition 9.** *Let* $\Pi_{\mathsf{FHE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *denote a homomorphic encryption scheme for a class of circuits* $\mathcal{C}$*. We say* $\Pi_{\mathsf{FHE}}$ *is* $\epsilon(\lambda)$*-circuit private if there is a (possibly unbounded) simulator* $\mathtt{Sim}$ *such that for all key-pairs* $(sk, pk)$ *in the support of* $\mathsf{KeyGen}(1^\lambda)$*, any circuit* $C \in \mathcal{C}$*, any plaintexts* $m_1, \ldots, m_n$ *and any fixed ciphertexts* $c_1, \ldots, c_n$ *such that* $c_i$ *in the image of* $\mathsf{Enc}_{pk}(m_i)$ *for each* $i \leq n$ *(i.e., each ciphertext* $c_i$ *is a valid encryption of the plaintext* $m_i$*) and any (unbounded) distinguisher* $\mathcal{D}$ *has advantage at most* $|p_{\mathcal{D},sim} - p_{\mathcal{D},Eval}| \leq \epsilon(\lambda)$ *where* $p_{\mathcal{D},sim} = \Pr[\mathcal{D}(sk, pk, c_1, \ldots, c_n, m_1, \ldots, m_n, \mathtt{Sim}(pk, C(m_1, \ldots, m_n)) = 1]$ *and* $p_{\mathcal{D},Eval} = \Pr[\mathcal{D}(sk, pk, c_1, \ldots, c_n, m_1, \ldots, m_n, \mathsf{Eval}_{pk}(C, c_1, \ldots, c_n)) = 1]$ *where the randomness is taken over the random coins of* $\mathcal{D}, \mathtt{Sim}$ *and* $\mathsf{Eval}$*.*

Since our distinguisher $\mathcal{D}$ is unbounded the requirement that the distinguishing advantage is upper bounded by $\epsilon(\lambda)$ is equivalent to requiring that the statistical distance between $\mathsf{Eval}_{pk}(C, c_1, \ldots, c_n)$ and $\mathtt{Sim}(pk, C(m_1, \ldots, m_n)$ is bounded by $\epsilon(\lambda)$. Giving the distinguisher additional information like $sk, pk, c_1, \ldots, c_n$ and $m_1, \ldots, m_n$ does not change the definition.

## D.3 Proving Conditional Encryption Secrecy

Theorem 13 proves that **Construction** 12 achieves conditional encryption secrecy as long as the underlying FHE scheme satisfies circuit privacy. Intuitively, the conditional encryption simulator will just use the circuit private FHE simulator. See details in the proof of Theorem 13.

**Theorem 13.** *Assume that* $\Pi_{\mathsf{FHE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is an* $\epsilon(\lambda)$*-circuit-private FHE, then* $\Pi_{\mathsf{CEnc}}$ *provides* $(\infty, t_{\mathtt{Sim}}, \epsilon(\lambda))$*-conditional encryption secrecy where* $t_{\mathtt{Sim}}$ *denotes the running time of the simulator for our circuit private FHE scheme.*

*Proof.* (Sketch) Supposing that $P(m_1, m_2) = 0$ we have $C_{P,m_2,m_3}(m_1) = Z$ where $Z$ is a fixed constant (publicly known). Thus, our conditional encryption secrecy simulator $\mathtt{Sim}_{CE}(pk)$ will simply run the FHE circuit privacy simulator $\mathtt{Sim}_{cpFHE}(pk, Z)$. Formally, we define $\mathtt{Sim}_{CE}(pk) \doteq (1, \mathtt{Sim}_{cpFHE}(pk, Z))$. Since $C_{P,m_2,m_3}(m_1) = Z$, by circuit privacy of the underlying FHE scheme, the output of $\mathtt{Sim}_{cpFHE}(pk, Z)$ will be statistically indistinguishable from the output of $\mathsf{CEnc}_{pk}(c_1, m_2, m_3) = (1, \mathsf{FHE.Eval}_{pk}(C_{P,m_2,m_3}, c_1))$. In particular, for any distinguisher $\mathcal{D}$ we have

$$\Big| \Pr\left[\mathcal{D}\left(sk, pk, c_1, m_1, m_2, m_3, \mathtt{Sim}_{cpFHE}\left(pk, C_{P,m_2,m_3}\left(m_1\right) = Z\right)\right) = 1\right] - \Pr\left[\mathcal{D}\left(sk, pk, c_1, m_1, m_2, m_3, \mathsf{Eval}_{pk}(C_{P,m_2,m_3}, c_1)\right) = 1\right] \Big| \leq \epsilon(\lambda) \tag{6}$$

By definition the distribution of $\mathtt{Sim}_{CE}(pk)$ its output is identical to $\mathtt{Sim}_{cpFHE}(pk, Z)$. Looking at **Construction 12**, we observe that $\mathsf{CEnc}_{pk}(c_1, m_2, m_3) = \mathsf{FHE.Eval}_{pk}(C_{P,m_2,m_3}, c_1)$. Thus we can rewrite Equation (6) as follows:

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

So we have shown the there exists simulator $\mathtt{Sim}_{CE}$ running in time $t_{\mathtt{Sim}}(\lambda) = t_{\mathtt{Sim}_{cpFHE}}(\lambda)$ such that any unbounded adversary can distinguish between $\mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ and $\mathtt{Sim}_{CE}(pk)$ with advantage of at most $\epsilon(\lambda)$ and **Construction 12** provides $(\infty, t_{\mathtt{Sim}}(\lambda), \epsilon(\lambda))$ conditional encryption secrecy.

□

## D.4 Correctness and Error Detection

**Theorem 14.** *Assuming that the underlying fully homomorphic scheme* FHE *satisfies perfect correctness* ***Construction 12*** *satisfies perfect correctness and perfect error detection i.e., the construction is* $1 - \epsilon(\lambda)$*-error detecting and* $1 - \epsilon(\lambda)$*-correct with* $\epsilon(\lambda) = 0$*.*

**Proof of Theorem 14:**

Correctness of **Construction 12** follows immediately from the correctness of the underlying FHE scheme. Observe that for any message $m$ in the message space and any secret key pair $(sk, pk) \leftarrow \Pi_{\mathsf{FHE}}.\mathsf{KeyGen}(1^\lambda)$ we have

$$\begin{aligned} \Pi_{\mathsf{CEnc}}.\mathsf{Dec}_{sk}\left(\Pi_{\mathsf{CEnc}}.\mathsf{Enc}_{pk}(m)\right) &= \Pi_{\mathsf{CEnc}}.\mathsf{Dec}_{sk}\left(0, \Pi_{\mathsf{FHE}}.\mathsf{Enc}_{pk}(m)\right) \\ &= \Pi_{\mathsf{FHE}}.\mathsf{Dec}_{sk}\left(\Pi_{\mathsf{FHE}}.\mathsf{Enc}_{pk}(m)\right) \\ &= m\ . \end{aligned}$$

The first equality holds by the definition of $\Pi_{\mathsf{CEnc}}.\mathsf{Enc}$. The second equality follows by correctness of the underlying FHE scheme and by the definition of $\Pi_{\mathsf{CEnc}}.\mathsf{Dec}_{sk}$ since we assumed the message $m$ is in message space for our conditional encryption scheme. The final equality follows by the correctness of the underlying fully homomorphic encryption scheme.

Similarly, consider any messages $m_1, m_2, m_3$ for which $P(m_1, m_2) = 1$ and $m_3$ is an arbitrary message in our message space. Let $(0, c_1) \leftarrow \Pi_{\mathsf{CEnc}}.\mathsf{Enc}_{pk}(m_1) = (0, \Pi_{\mathsf{FHE}}.\mathsf{Enc}_{pk}\left(m_1\right))$ be any encryption of $m_1$. By construction we have

$$\Pi_{\mathsf{CEnc}}.\mathsf{CEnc}_{pk}\left((0, c_1), m_2, m_3\right) = \left(1, \Pi_{\mathsf{FHE}}.\mathsf{Eval}_{pk}\left(C_{P,m_2,m_3}, c_1\right)\right)\ ,$$

and by correctness of $\Pi_{\mathsf{FHE}}$ we have

$$\Pi_{\mathsf{FHE}}.\mathsf{Dec}_{sk}\left(\Pi_{\mathsf{FHE}}.\mathsf{Eval}_{pk}\left(C_{P,m_2,m_3}, c_1\right)\right) = C_{P,m_2,m_3}(m_1) = m_3\ ,$$

where the first equality follows by FHE correctness — note that $c_1 = \Pi_{\mathsf{FHE}}.\mathsf{Enc}_{pk}(m_1)$. The second equality follows by definition of $C_{P,m_2,m_3}$ since, by assumption, we have $P(m_1, m_2) = 1$. Since we assume that $m_3$ is in our message space by definition of $\Pi_{\mathsf{CEnc}}.\mathsf{Dec}$ it follows that

$$\Pi_{\mathsf{CEnc}}.\mathsf{Dec}_{sk}\left(\Pi_{\mathsf{CEnc}}.\mathsf{CEnc}_{pk}\left(\left(0, c_1\right), m_2, m_3\right)\right) = m_3\ .$$

Our argument that **Construction 12** satisfies perfect error detection also follows from the correctness of the underlying FHE scheme. Let $m_1$ and $m_2$ be any messages in the message space for which the predicate does not hold i.e., $P(m_1, m_2) = 0$. Let $(0, c_1) \leftarrow \Pi_{\mathsf{CEnc}}.\mathsf{Enc}_{pk}(m_1) = (0, \Pi_{\mathsf{FHE}}.\mathsf{Enc}_{pk}(m_1))$ be any encryption of $m_1$. By construction we have

$$\Pi_{\mathsf{CEnc}}.\mathsf{CEnc}_{pk}\left((0, c_1), m_2, m_3\right) = \Pi_{\mathsf{FHE}}.\mathsf{Eval}_{pk}(C_{P,m_2,m_3}, c_1)\ ,$$

and by correctness of $\Pi_{\mathsf{FHE}}$ we have

$$\Pi_{\mathsf{FHE}}.\mathsf{Dec}_{sk}\left(\Pi_{\mathsf{FHE}}.\mathsf{Eval}_{pk}\left(C_{P,m_2,m_3}, c_1\right)\right) = C_{P,m_2,m_3}(m_1) = Z \ ,$$

where $Z$ is a message explicitly chosen to be outside the message space of our conditional encryption scheme. The first equality above follows by $\mathsf{FHE}$ correctness since $c_1 = \Pi_{\mathsf{FHE}}.\mathsf{Enc}_{pk}(m_1)$ and the second equality follows by definition of $C_{P,m_2,m_3}$ since, by assumption, we have $P(m_1, m_2) = 0$. Thus, by definition of $\Pi_{\mathsf{CEnc}}.\mathsf{Dec}$ it follows that

$$\Pi_{\mathsf{CEnc}}.\mathsf{Dec}_{sk}\left(\Pi_{\mathsf{CEnc}}.\mathsf{CEnc}_{pk}\left((0, c_1), m_2, m_3\right)\right) = \perp \ .$$

□

# E Missing Proofs

**Reminder of Theorem 1.** *Construction 18 is a perfectly correct and $1-\epsilon(\lambda)$-error detecting conditional encryption scheme with $\epsilon(\lambda) = \frac{|\Sigma|^{n+1}}{N} \leq \frac{1}{\max\{p,q\}}$.*

**Proof of Theorem 1:** Since $\mathsf{Enc}_{pk}(m)$ simply runs regular Pallier encryption perfect correctness of Pallier immediately implies that $\mathsf{Dec}_{sk}\left(\mathsf{Enc}_{pk}(m)\right) = \mathsf{Dec}_{sk}\left(P.\mathsf{Enc}_{pk}(\mathsf{ToInt}(m)\right) = \mathsf{ToInt}^{-1}\left(\mathsf{ToInt}(m)\right) = m$ with probability 1 for all messages $m \in \Sigma^{\leq n}$ and all public/private key pairs in the support of $\mathsf{KeyGen}$. Similarly, if $c_1 = \mathsf{Enc}_{pk}(m)$ and $P_{=}(m_1, m_2) = 1$ then $\mathsf{CEnc}_{pk}(c_1, m_2, m_3)$ will output $(1, c')$ where $c' = g^{\mathsf{ToInt}(m_3)} r^n \mod N^2$ for some $r \in \mathbb{Z}_N^*$. Thus, $c'$ is a valid pallier ciphertext for $\mathsf{ToInt}(m_3)$ and, by correctness of Pallier, $\mathsf{Dec}_{sk}(1, c')$ will return $m_3$.

On the other hand if $P_{=}(m_1, m_2) = 0$ then by Theorem 2 the ciphertext $c'$ is a valid Pallier Ciphertext for some uniformly random integer $y \in \mathbb{Z}_N$ and we will have $\mathsf{Dec}_{sk}(1, c') = \perp$ as long as $y > |\Sigma|^{n+1}$. Thus, the construction is $1 - \epsilon(\lambda)$-error detecting conditional encryption scheme with $\epsilon(\lambda) = \frac{|\Sigma|^{n+1}}{N} \leq \frac{1}{\max\{p,q\}}$. □

**Reminder of Theorem 2.** *The conditional encryption scheme described in Construction 18 provides $(\infty, t_{\mathtt{Sim}}, 0)$ conditional encryption secrecy in which $t_{\mathtt{Sim}} = t_{\mathtt{P.Enc}}$ is time of doing one Paillier Encryption.*

**Proof of Theorem 2:** We define the simulator $\mathtt{Sim}(pk)$ as follows. The simulator $\mathtt{Sim}(pk)$ takes as input the Paillier public key $pk$ and then selects $R_s \in_R \mathbb{Z}_N$ and $r_s \in_R \mathbb{Z}_N^*$ uniformly at random and then encrypts $R_s$ as $C_{\mathtt{Sim}} = \mathtt{P.Enc}_{pk}(R_s; r_s) = g^{R_s} r_s^N \mod N^2$ and outputs it. We now argue that for any $m_1, m_2 \in \Sigma^n$ with $P_{=}(m_1, m_2) = 0$ any payload message $m_3$ and any Pallier key $(pk = (N = pq, g), sk)$ which satisfies our condition that $|\Sigma|^{n+1} \leq \min\{p, q\}$ and any encryption $c_1 = g^{m_1} r_1^N \mod N^2$ of $m_1$ under $pk$ that the distributions $(pk, sk, m_1, m_2, m_3, c_1, C_{m_3} = \mathsf{CEnc}_{pk}\left(c_{m_1}, m_2, m_3\right))$ and $(pk, sk,\ m_1, m_2, m_3, c_1, C_{\mathtt{Sim}} = \mathtt{Sim}(pk))$ are identical. In particular, it suffices to argue that $C_{\mathtt{Sim}} = \mathtt{Sim}(pk)$ and $\mathsf{CEnc}_{pk}(c_{m_1}, m_2, m_3)$ are distributed identically.

To see this consider the generation of $\mathsf{CEnc}_{pk}(c_{m_1}, m_2, m_3)$. First, we pick a random $R \in \mathbb{Z}_N$ and generate an encryption of $(-R \cdot m_2 \mod N)$ as $c_2 = g^{-R \cdot m_2} r_2^N \mod N^2$ where $r_2 \in \mathbb{Z}_N^*$ is picked randomly. We then compute $c_1^R = g^{m_1 \cdot R} r_1^{RN}$. Finally, we output

$$\begin{aligned} &c_1^R c_2 \cdot g^{m_3} \mod N^2 = g^{m_3 + R(m_1 - m_2)} r_1^{RN} r_2^N \\ &= g^{m_3 + R(m_1 - m_2) \mod N} \left(r_1^R r_2 \mod N\right)^N \mod N^2 \ . \end{aligned}$$

where the values $R \in \mathbb{Z}_N$, $r_2 \in \mathbb{Z}_N^*$ are fresh random values. In the last step we implicitly used the fact that if $r_1^R r_2 = aN + b$ where $b = \left[r_1^R r_2 \mod N\right]$ then

$$(aN + b)^N = \sum_{i=0}^{N} \binom{N}{i} (aN)^i b^{N-i} = b^N \mod N^2 \ .$$

Let us first focus on the term $m_3 + R(m_1 - m_2)$ in the exponent of $g$. We observe that $[m_1 - m_2 \mod N] \in \mathbb{Z}_N^*$ since $1 \leq |m_1 - m_2| \leq |\Sigma|^n \leq \min\{p, q\}$. It follows that for any $m_3$ that $R(m_1 - m_2) + m_3$ is distributed uniformly at random in $\mathbb{Z}_N$ when $R \in \mathbb{Z}_N$ is picked randomly. We next consider the term $\left(r_1^R r_2 \mod N\right)$ and argue for any fixed $r_1 \in \mathbb{Z}_N^*$ and $R \in \mathbb{Z}_N$ that $\left(r_1^R r_2 \mod N\right)$ is distributed uniformly at random in $\mathbb{Z}_N^*$ when $r_2 \in \mathbb{Z}_N^*$ is picked randomly. It follows that for any $r_1 \in \mathbb{Z}_N^*, m_1, m_2, m_3$ such that $1 \leq |m_1 - m_2| \leq \min\{p, q\}$ that the simulated ciphertext ($C_{\mathtt{Sim}} = \mathtt{Sim}(pk) = g^{R_s} r_s^N \mod N^2$ for random $r_s \in \mathbb{Z}_N^*$ and $R_s \in \mathbb{Z}_N$ ) is identically distributed to $\mathsf{CEnc}_{pk}(c_{m_1}, m_2, m_3)$. □

**Reminder of Theorem 3.** *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 $\mathcal{D}_{ak}$ which outputs random values between $0, \cdots, ak$. Then the statistical distance between these two distributions is $\mathtt{SD}(\mathcal{D}_{ak}, \mathcal{U}_b) = \frac{r}{b} \leq \frac{1}{k+1}$.*

**Proof of Theorem 3:** Based on the definition of statistical distance we have

$$\begin{aligned}
\mathtt{SD}(\mathcal{D}_{ak}, \mathcal{U}_b) &= \frac{1}{2} \sum_{i=0}^{b-1} \left| \Pr_{y \in_R \mathcal{D}_{ak}}[y = i] - \Pr_{y \in_R \mathcal{U}_b}[y = i] \right| \\
&= \frac{1}{2}(ak)\left(\left|\frac{1}{b} - \frac{1}{a} \cdot \frac{1}{k}\right| + (b - ka)\left(\left|\frac{1}{b} - 0\right|\right)\right. \\
&= \frac{1}{2}(ak)\left(\frac{b - ak}{abk}\right) + \frac{1}{2}(r)\left(\frac{1}{b}\right) = \frac{r}{b} \leq \frac{1}{k+1} \qquad (7)
\end{aligned}$$

□

**Theorem 15.** ***Construction 19*** *is a $1 - \epsilon(\lambda)$-correct and a $1 - \epsilon(\lambda)$-error detecting conditional encryption scheme with $\epsilon(\lambda) = \binom{n}{\ell}\epsilon_{AE}(\lambda) + 2^{-\lambda}$. Here,*

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

*denotes the negligible probability that the ciphertext $c = \mathsf{Enc}_{K_2}(m)$ is still valid under an unrelated key $K_1$.*

**Proof of Theorem 15:** 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 the Definition 2 which need to be proved. The first condition is regular encryption correctness and the other one is the conditional encryption correctness.

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 key generation algorithm follows immediately from the correctness of Pallier encryption.

It remains to to show that for all messages $m_1, m_2 \in \Sigma^n$ such that $P_{\ell,\mathtt{Ham}}(m_1, m_2)$, all payload messages $m_3$, all $\{sk, pk\}$ in the support of our Key Generation algorithm and all random strings $r_1, r_2 \in_R (\mathbb{Z}_N^*)^n$ we have

$$\Pr\left[\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(c_1, m_2, m_3; r_2)) = m_3 \,\middle|\, \begin{array}{c}(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_1 = \mathsf{Enc}_{pk}(m_1; r_1) \\ P_{t,\mathtt{Ham}}(m_1, m_2) = 1\end{array}\right] \geq 1 - \epsilon\ . \qquad (8)$$

Let $K$ denote the authenticated encryption key and let $[\![s]\!]_1, \ldots, [\![s]\!]_n$ denote the shares of $K$ that were generated by the conditional encryption algorithm. Let $\tilde{c} = (b, \tilde{c}_1, \ldots, \tilde{c}_n, C_{AE})$ denote the output of $\mathsf{CEnc}_{pk}(c_1, m_2, m_3; r_2))$, and let $[\![s']\!]_i = \mathsf{RDec}\left(P.\mathsf{Dec}_{sk}(\tilde{c}_i)\right)$ denote the shares that are recovered. Finally, let $S^* = \{i \in [n] \ : \ m_2[i] = m_1[i]\}$ denote indices of the characters where $m_2$ and $m_1$ match. By correctness of Pallier we have $[\![s']\!]_i = [\![s]\!]_i$ for *all* $i \in S^*$. For $i \notin S^*$ the distribution over $[\![s']\!]_i$ is as follows: sample a uniformly random item $y_i$ from $\mathbb{Z}_N^*$ and output $y_i \mod 2^\lambda$.

If $P_{Hamm,\ell}(m_1, m_2) = 1$ we have $|S^*| \geq n - \ell$ and there is some subset $S \subseteq S^*$ of size $|S| = n - \ell$ such that

$$K = K_S = \mathsf{SecretRecover}\left(\left\{(i, [\![s']\!]_i)_{i \in S}\right\}\right) .$$

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

Thus, the only possible to output an incorrect message $m'$ is if for some $S \subseteq n$ of size $n - \ell$ we have $K \neq K_S = \mathsf{SecretRecover}\left(\left\{(i, [\![s']\!]_i)_{i \in S}\right\}\right)$ and $\mathsf{Auth.Dec}_{K_{S^*}}(c_{AE}) \neq \perp$. However, if $K_S \neq K$ then $S \not\subseteq S^*$ and we can find some $i \in S \setminus S^*$. For now assume that for all $i \notin S^*$ the value of $[\![s']\!]_i$ is uniformly random we can view $K_S$ as a uniformly random key. If we view each $K_S$ as random then we have $\Pr[\mathsf{Auth.Dec}_{K_S}(c_{AE}) \neq \perp] \leq \epsilon_{AE}$ and $\Pr[\exists S \subseteq [n] \ .\mathsf{Auth.Dec}_{K_S}(c_{AE}) \notin \{m_3, \perp\}] \leq \binom{n}{\ell}\epsilon_{AE}$.

In the previous paragraph we assumed that the value $[\![s']\!]_i$ is uniformly random for each $i \notin S^*$ the value. This is close, but it is not quite true. In reality the distribution of $[\![s']\!]_i$ is described by sampling a uniformly random $y_i \in \mathbb{Z}_N^*$ and then outputting $y_i \mod 2^\lambda$.However, by Theorem 3 the statistical distance between original/modified distribution of our recovered shares $[\![s']\!]_1, \ldots, [\![s']\!]_n$ is upper bounded by $2^{-\lambda}$. This follows since we are guaranteed that $N > n2^{2\lambda}$ by definition of the key generation algorithm. Thus, we have

$$\Pr\left[\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(c_1, m_2, m_3; r_2)) \neq m_3 \middle| \begin{array}{c}(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_1 = \mathsf{Enc}_{pk}(m_1; r_1) \\ P_{t,\mathtt{Ham}}(m_1, m_2) = 1\end{array}\right] \leq \binom{n}{\ell}\epsilon_{AE} + 2^{-\lambda} .$$

Similarly, if $P_{\ell,Hamm}(m_1, m_2) = 0$ then for all $S \subseteq [n]$ of size $|S| = n - \ell$ we can (essentially) view $K_S$ as random since there is some $i \in S \setminus S^*$. It follows that

$$\Pr\left[\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(c_1, m_2, m_3; r_2)) \neq \perp \middle| \begin{array}{c}(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c_1 = \mathsf{Enc}_{pk}(m_1; r_1) \\ P_{t,\mathtt{Ham}}(m_1, m_2) = 0\end{array}\right] \leq \binom{n}{\ell}\epsilon_{AE} + 2^{-\lambda} .$$

□

**Reminder of Theorem 4.** *[Conditional Encryption Secrecy of **Construction 19**] 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 $\lambda$ and any running time parameter $t_{AE}$. Then for any $t$ and any security parameter $\lambda$ **Construction 19** provides $(t, t_{\mathtt{Sim}}, \epsilon(t, \lambda))$ conditional encryption secrecy with $\epsilon(t, \lambda) \leq \epsilon_{AE}(t, \lambda) + 2^{-\lambda}$ and $t_{\mathtt{Sim}} = n \cdot t_{P.\mathsf{Enc}} + \mathtt{poly}(\lambda)$.*

**Proof of Theorem 4:** To prove this theorem we use a hybrid argument. In the first hybrid (Hybrid 0, real world) the distinguisher is given the actual ciphertext output conditional encryption and in the last hybrid contains the adversary is given a ciphertext output by our simulator — described in Figure 3. As the hybrids are indistinguishable, we can conclude that the first and last hybrid are indistinguishable as well which implies that the our suggested construction is secure and provides conditional encryption secrecy in the semi-honest model. Then we concretely compute the distinguishing advantage of the defined hybrids. In what follows, we describe the hybrids with more details.

- **Hybrid 0**: In this hybrid the distinguisher $\mathcal{D}$ is given $(sk, pk, m_1, , m_2, \ m_3, c_{m_1}, (1, \tilde{c}))$ 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 $T = \{i : m_2[i] \neq m_1[i]\}$ be the set of the indexes that $m_1$ and $m_2$ have different characters. We define **Hybrid 1** similar to **Hybird 0**, except for all $j \in T$ we replace

  $$\tilde{c}_j = P.\mathsf{Enc}_{pk}\Big(R_i(m_2[j] - m_1[j]) + \mathsf{REnc}([\![s]\!]_i)\Big)$$

  with $P.\mathsf{Enc}(R_j)$ where $R'_j \in_R \mathbb{Z}_N$ are fresh and uniform random values chosen from $\mathbb{Z}_N$.
- **Hybrid 2**: This hybrid is exactly the same as the previous hybrid except we replace all the remaining ciphertexts $j \in [1 : n]/T$ with $\tilde{c}_j = P.\mathsf{Enc}_{pk}\Big(R_j(m_2[j] - m_1[j]) + \mathsf{REnc}([\![s_r]\!]_j)\Big)$ where $[\![s_r]\!]_j \in_R \{0,1\}^\lambda$ are fresh uniformly random elements (chosen independently from the secret $K$) chosen from the field $\mathbb{F}_{2^\lambda}$.

- **Hybrid 3**: This hybrid is exactly the same as the previous hybrid except we replace all the ciphertexts $j \in [1:n]/T$ with $\tilde{c}_j = P.\mathsf{Enc}_{pk}\Big(R_j(m_2[j] - m_1[j]) + \hat{R}_j)\Big)$ where $\hat{R}_j \in_R \mathbb{Z}_N$ are chosen from $\mathbb{Z}_N$ uniformly at random.
- **Hybrid 4**: This hybrid is exactly the same as the previous hybrid except we replace ciphertexts $\tilde{c}_j$ for all $j \in [1:n]/T$, with $\mathtt{P}.\mathsf{Enc}_{pk}(R'_j)$ in which $R'_j \in_R \mathbb{Z}_N$ are chosen from $\mathbb{Z}_N$ uniformly at random.
- **Hybrid 5**: This hybrid is exactly the same as the previous hybrid unless we replace $c_{AE}$ with $c'_{AE} \in_R \{0,1\}^{l(\lambda)}$ a $\lambda$-bit string chosen uniformly at random. We note that $l(\lambda)$ is a polynomila over the security parameter $\lambda$ which represents the ciphertext size of authenticated encryption.
- **Hybrid 6**: We replace the ciphertext of the conditional encryption with the output of the simulator $\mathtt{Sim}$ described in Figure 3.

Now we are proving that the defined hybrids are equivalent.

#### E.0.1 Hybrid 0 ≡ Hybrid 1

These hybrids are equivalent i.e., we have

$$\Pr[\mathcal{D}^{H_0} = 1] = \Pr[\mathcal{D}^{H_1} = 1] \ . \tag{9}$$

Where $\mathcal{D}^{H_i} = 1$ denotes the event that the distinguisher outputs 1 in hybrid $i$. The argument is essentially the same as what we had for the security of *Equality test* predicate — see the proof of Theorem 2. In particular, for each $j \in T$ we have $m_1[j] \neq m_2[j]$ and $|m_1[j] - m_2[j]| \leq \min\{p,q\}$ which implies that $(m_1[j] - m_2[j]) \in \mathbb{Z}_N^*$. It follows that $R_j \times (m_1[j] - m_2[j])$ is uniformly random in $\mathbb{Z}_N$.

#### E.0.2 Hybrid 1 ≡ Hybrid 2

We have information theoretically eliminated all information about shares $\mathsf{shares}$ with $j \in T$. Since $P_{\ell,\mathtt{Ham}}(m_1, m_2) = 0$ we have $|T| > \ell$ and $|\overline{T}| < n - \ell$. Let $T = \{i_1, \ldots, i_t\}$ with $t < n - \ell$. Shamir Secret Sharing guarantees that $(s_{i_1}, s_{i_2}, \ldots, s_{i_t})$ is uniformly random in $\mathbb{F}_{2^\lambda}^t$. Thus, we can simply replace the shares with uniformly random values. We have

$$\Pr[\mathcal{D}^{H_1}] = \Pr[\mathcal{D}^{H_2}] \ . \tag{10}$$

#### E.0.3 Statistically indistinguishability of Hybrid 2 ≡ Hybrid 3

We apply Theorem 3 with $a = 2^\lambda$, $k = \lfloor \frac{N}{2^\lambda} \rfloor$ and $b = N$. We first observe that when $i \in \overline{T}$ the value of $s_i \in \mathbb{F}_{2^\lambda}$ is uniformly random so that $\mathsf{REnc}(s_i)$ is equivalent to $\mathcal{D}_{ak}$. It follows that the statistical distance between $\mathsf{REnc}(s_i)$ and the uniform ditribution $\mathbb{Z}_N$ is at most $\frac{1}{k} = \lfloor \frac{N}{2^\lambda} \rfloor^{-1}$. Since we are replacing the random value in $|\overline{T}|$ ciphertexts the overall statistical distance is upper bounded by $\frac{|\overline{T}|}{k} \leq \frac{n}{k}$ we have:

$$|\Pr[\mathcal{D}^{H_2}] - \Pr[\mathcal{D}^{H_3}]| \leq \frac{2^\lambda n}{N - 2^\lambda} \leq 2^{-\lambda} \ . \tag{11}$$

The last inequality follows since we pick $N \geq 2n2^{2\lambda}$ so that $\frac{2^\lambda n}{N - 2^\lambda} \leq 2^{-\lambda}$.

#### E.0.4 Hybrid 3 ≡ Hybrid 4

These hybrids are statistically indistinguishable as $R_j(m_2[j] - m_1[j]) + \hat{R}_j$ is already uniformly random in $\mathbb{Z}_N$. We have

$$\Pr[\mathcal{D}^{H_3}] = \Pr[\mathcal{D}^{H_4}] \tag{12}$$

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

1. Sample, $r''_1, \ldots, r''_n \in_R \mathbb{Z}^*_N, R''_1, \ldots, R''_n \in_R \mathbb{Z}_N{}^n$ uniformly at random
2. For all $1 \leq i \leq n$ 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 $\tilde{c}' = (1, \tilde{c}'_1, \cdots, \tilde{c}'_n, c'_{AE})$.

Figure 3: Steps of designing the simulator $\mathtt{Sim}$ for the conditional encryption secrecy when the predicate is $P_{\ell,\mathtt{Ham}}$

### E.0.5 Indistinguishability of Hybrid 4 and Hybrid 5

By Hybrid 4 we have information theoretically elimated 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 $\epsilon_{AE}(t_{AE}, \lambda)$. So we have

$$|\Pr[\mathcal{D}^{H_4}] - \Pr[\mathcal{D}^{H_1} = 1]| \leq \epsilon_{AE}(t_{AE}, \lambda) \tag{13}$$

### E.0.6 Hybrid 5 ≡ Hybrid 6

Looking at the definition of our our simulator in Figure 3, 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

$$\Pr[\mathcal{D}^{H_5}] = \Pr[\mathcal{D}^{H_6}] \tag{14}$$

Putting everything together we have

$$\begin{aligned}
&\Big| \Pr\left[\mathcal{D}\left(sk, pk, m_1, m_2, m_3.\mathsf{CEnc}_{pk}\left(\mathsf{Enc}_{pk}\left(m_1\right), m_2\right)\right) = 1\right] \\
&- \Pr\left[\mathcal{D}\left(sk, pk, m_1, m_2, m'_3, \mathtt{Sim}\left(pk\right)\right) = 1\right] \Big| \\
&= \left|\Pr\left[\mathcal{D}^{H_0}\right] - \Pr\left[\mathcal{D}^{H_6}\right]\right| \\
&\leq \sum_{i=0}^{5} \left|\Pr\left[\mathcal{D}^{H_i}\right] - \Pr\left[\mathcal{D}^{H_{i+1}}\right]\right| \\
&< \epsilon_{AE}(t', \lambda) + \frac{n2^\lambda}{N - 2^\lambda} \leq \epsilon_{AE}(t', \lambda) + 2^{-\lambda} \ .
\end{aligned}$$

□

**Reminder of Theorem 5.** *$\Pi_{1,\mathtt{ED}}$ is a $1 - \epsilon(\lambda)$ correct conditional encryption scheme for the predicate $P_{1,\mathtt{ED}}$ and $\Pi_{1,\mathtt{ED}}$ is $1 - \epsilon(\lambda)$-error detecting with $\epsilon(\lambda) = \frac{(2n+1)|\Sigma|^{n+1}}{N} \leq \frac{2n+1}{\max\{p,q\}}$.*

**Proof of Theorem 5:** Note that $\mathsf{Enc}_{pk}(m)$ includes $c[0] = P.\mathsf{Enc}_{pk}(\mathsf{ToInt}(m))$ and that therefore by correctness of Pallier we have $\mathsf{Dec}_{sk}\left(\mathsf{Enc}_{pk}(m)\right) = \mathsf{Dec}_{sk}\left(P.\mathsf{Enc}_{pk}(\mathsf{ToInt}(m))\right) = \mathsf{ToInt}^{-1}\left(\mathsf{ToInt}(m)\right) = m$ with probability 1 for all messages $m \in \Sigma^{\leq n}$ and all public/private key pairs in the support of $\mathsf{KeyGen}$.

Recall that if $c = (0, c[0], \ldots, c[n]) = \mathsf{Enc}_{pk}(m)$ then $\mathsf{CEnc}_{pk}(c, m', m'')$ will output a ciphertext of the form $(1, \tilde{c}_0, \ldots, \tilde{c}_{2n})$. If $P_{1,\mathtt{ED}}(m, m') = 0$ then we have $m_{-j} \neq m'$ and $m \neq m'_{-j}$ for all $0 \leq j \leq n$. Thus, by Theorem 2 each $\tilde{c}_j = g^{y_j} r_j^n \mod N^2$ for random values $r_j \in \mathbb{Z}_N^*$ and $y_j \in \mathbb{Z}_N$. Thus, we have

$$\Pr[\mathsf{Dec}_{sk}(1, \tilde{c}_0, \ldots, \tilde{c}_{2n}) \neq \perp] \leq \Pr[\exists j.y_j < |\Sigma|^{n+1}] \leq \frac{(2n+1)|\Sigma|^{n+1}}{N} .$$

This implies that the construction is $1 - \epsilon(\lambda)$-error detecting conditional encryption scheme with $\epsilon(\lambda) = \frac{|\Sigma|^{n+1}}{N} \leq \frac{1}{\max\{p,q\}}$.

Finally, if $P_{1,\mathtt{ED}}(m, m') = 0$ then by perfect correctness of our conditional encryption scheme for equality predicate there exists some $j$ such that $\tilde{c}_j = g^{y_j} r_j^N \mod N^2$ is a valid Pallier encryption of $y_j = \mathsf{ToInt}(m'')$. Furthermore, we have already shown that $\Pr[\exists j.y_j < |\Sigma|^{n+1} \wedge y_j \neq \mathsf{ToInt}(m'')] \leq \frac{(2n+1)|\Sigma|^{n+1}}{N}$. It follows that, except with probability $\frac{(2n+1)|\Sigma|^{n+1}}{N}$ that we will have $\mathsf{Dec}_{sk}(1, \tilde{c}_0, \ldots, \tilde{c}_{2n}) = \mathsf{ToInt}^{-1}(\mathsf{ToInt}(m''))$. □

**Reminder of Theorem 7.** *Suppose that we are given $k$ separate conditional encryption schemes $\Pi_1, \ldots, \Pi_k$ corresponding predicates $P_1, \ldots, P_k$ and that each $\Pi_i$ provides $(t(\lambda), t_{\mathtt{Sim},i}(\lambda), \epsilon_i(t(\lambda), \lambda))$-conditional encryption secrecy. The construction $\Pi_{or}$ provides $(t'(\lambda), t'_{\mathtt{Sim}}(\lambda), \epsilon'(t'(\lambda), \lambda))$-conditional encryption secrecy with $t'(\lambda) = O(t(\lambda))$, $t'_{\mathtt{Sim}}(\lambda) \approx \sum_{i=1}^k t_{\mathtt{Sim}_i}(\lambda)$ and $\epsilon'(t'(\lambda), \lambda) = \sum_i^k \epsilon_i(t'(\lambda), \lambda)$.*

**Proof of Theorem 7:** The simulator $\mathtt{Sim}_{OR}(pk)$ for $\Pi_{OR}$ will run the simulator $\mathtt{Sim}_i(pk_i)$ for each conditional encryption scheme [9] and concatenate all of the ciphertexts. Clearly, the running time of the simulator is $t'_{\mathtt{Sim}}(\lambda) \approx \sum_{i=1}^k t_{\mathtt{Sim},i}(\lambda)$. We can now define a sequence of $k+1$ hybrids Hybrid 0 to Hybrid $k$. Intuitively, in hybrid $i$ we set $c_j = \mathtt{Sim}_i(pk)$ for $j \leq i$ and $c_j = \Pi_j.\mathsf{CEnc}(c, m', m'')$ for $j > i$. Note that in Hybrid 0 we have $c_j = \Pi_j.\mathsf{CEnc}(c, m', m'')$ for all $j$ and thus the final output is $\mathsf{CEnc}(c, m', m'')$. By contrast, in Hybrid $k$ we have $c_j = \mathtt{Sim}_j(pk_j)$ for all $j \leq k$ and thus the final output is $\mathtt{Sim}_{OR}(pk)$.

By assumption any attacker running in time $t'(\lambda) = t(\lambda) - o(t(\lambda))$ can distinguish hybrids $i-1$ and $i$ with probability at most $\epsilon_i(t(\lambda), \lambda)$. It follows that any attacker running in time $t'(\lambda) = t(\lambda) - o(t(\lambda))$ can distinguish hybrid 0 from hybrid $k$ with probability at most $\epsilon'(\lambda, t'(\lambda)) = \sum_{i=1}^k \epsilon_i(t(\lambda), \lambda)$. □

**Reminder of Theorem 8.** *Suppose that we are given $k$ separate conditional encryption schemes $\Pi_1, \ldots, \Pi_k$ corresponding to predicates $P_1, \ldots, P_k$ and that each $\Pi_i$ is $1 - \epsilon_i(\lambda)$-correct and $1 - \epsilon'_i(\lambda)$-error detecting. Then the construction $\Pi_{or}$ is $1 - \epsilon(\lambda)$-correct (resp. $1 - \epsilon'(\lambda)$-error detecting) with $\epsilon(\lambda) = \sum_i^k \epsilon'_i(\lambda) + \sum_i^k \epsilon_i(\lambda)$ (resp. $\epsilon'(\lambda) = \sum_{i=1}^{\lambda} \epsilon'_i(\lambda)$).*

**Proof of Theorem 8:** Let $T = \{j : P_j(m_1, m_2) = 1\}$ and $\overline{T} = \{j : P_j(m_1, m_2) = 0\} = [k] \setminus T$. We first suppose that $P_{OR}(m_1, m_2) = 0$ which implies that $P_i(m_1, m_2) = 0$ for all $i \leq k$ i.e., $\overline{T} = [k]$.

Now let $(pk, sk)$ be any honestly generated public/secret key and let $c = (0, c_1, \ldots, c_k) = \mathsf{CEnc}_{pk}(m_1)$ with $c_i \doteq \Pi_i.\mathsf{Enc}_{pk}(m_1)$. The probability that $\Pi_i.\mathsf{Dec}_{sk}(\Pi_i.\mathsf{CEnc}_{pk}(c_i, m_2, m_3)) \neq \perp$ is at most $\epsilon'_i(\lambda)$. Union bounding over all $i \leq k$ the probability that there exists $i$ such that $\Pi_i.\mathsf{Dec}_{sk}(\Pi_i.\mathsf{CEnc}_{pk}(c_i)) \neq \perp$ is at most $\sum_{i=1}^k \epsilon'_i(\lambda) = \epsilon'(\lambda)$.

On the other hand suppose that $P_{OR}(m_1, m_2) = 1$ which means that $P_j(m_1, m_2) = 1$ for some $j \leq k$. Clearly, if $|T| \geq 1$ and $\Pi_i.\mathsf{Dec}_{sk}(\Pi_i.\mathsf{CEnc}_{pk}(c_i)) = m_3$ for all $i \in T$ and $\Pi_i.\mathsf{Dec}_{sk}(\Pi_i.\mathsf{CEnc}_{pk}(c_i)) = \perp$ for all $i \notin T$ then $\mathsf{Dec}_{sk}$ will output the correct message $m_3$. As before the probability that there exists $i \in \overline{T}$ such that $\Pi_i.\mathsf{Dec}_{sk}(\Pi_i.\mathsf{CEnc}_{pk}(c_i)) \neq \perp$ is at most $\sum_{i=1}^k \epsilon'_i(\lambda) = \epsilon'(\lambda)$. Similarly, the probability that there exists $j \in T$ such that $\Pi_i.\mathsf{Dec}_{sk}(\Pi_i.\mathsf{CEnc}_{pk}(c_i)) \neq m_3$ is at most $\sum_{j=1}^k \epsilon_i(\lambda)$.

Thus, we have $\Pr[\mathsf{Dec}_{sk}(\mathsf{CEnc}_{pk}(\mathsf{Enc}_{pk}(m_1), m_2, m_3)) \neq m_3] \leq \sum_{i=1}^k (\epsilon'_i(\lambda) + \epsilon_i(\lambda)) = \epsilon(\lambda)$.

□

[9] In the malicious security setting the simulator $\mathtt{Sim}_{OR}(b, pk)$ is also given a bit $b = 1$ if and only if $\mathsf{CEnc}_{pk}(c, m', m'') = \perp$ i.e., if and only if $\Pi_i.\mathsf{CEnc}_{pk}(c, m', m'') = \perp$ for some $i \leq k$. If $b = 1$ then $\mathtt{Sim}_{OR}(b, pk)$ outputs $\perp$. Otherwise we simply run $\mathtt{Sim}_i(0, pk_i)$ for each $i \leq k$.

# F System Model of Personalized Typo Tolerance

In this section, we will concentrate on the application of the conditional encryption scheme in designing a secure mechanism to supporting password typos in a secure way. We first start with describing the syntax and API of a typical password-based authentication server, and then we use our introduced conditional encryption schemes for handling the typos when the user logs in with a wrong password close the original one. Indeed we expect the if the miss-typed password distance from the original one is *small*, then this typo can be used by the user for future logins.

A password-based authentication scheme $\Pi$ is a stateful mechanism that includes three main algorithms: $\mathsf{Initialization}$, $\mathsf{RegisterNewUser}$ and $\mathsf{Login}$ which are described as follows.

- $(\sigma_0) \leftarrow \mathsf{Initialization}(1^\lambda)$: The initialization algorithm (potentially randomized) $\mathsf{Initialization}$ takes as input the security parameter $\lambda$ and setups the system and outputs the initial state $\sigma_0$.
- $(\sigma', d \in \{\mathtt{acpt}, \mathtt{rjct}\}) \leftarrow \mathsf{RegisterNewUser}(\sigma, \mathtt{Id}, \mathtt{pwd})$: This is potentially a randomized algorithm which takes as input the current state of the system $\sigma$, the user identity $\mathtt{Id}$ and its corresponding password $\mathtt{pwd}$, and updates the state of the system to $\sigma'$ and output $\mathtt{acpt}$ (for the successful registration when the user id $\mathtt{Id}$ is new and previously is not registered) or $\mathtt{rjct}$ (for unsuccessful registration).
- $(\sigma', d \in \{\mathtt{acpt}, \mathtt{rjct}\}) \leftarrow \mathsf{Login}(\sigma', \mathtt{Id}, \mathtt{pwd})$: Let $\sigma$ be the current state. This is also potentially is a randomized algorithm which takes as input the state $\sigma$, the identity $\mathtt{Id}$ and password $\mathtt{pwd}$, and outputs the updated state $\sigma'$ and the login result $d$. It either outputs $d = \mathtt{acpt}$ (for successful login) or $d = \mathtt{rjct}$ (unsuccessful login attempt e.g., using wrong password).

In what follows, we will formally define the required properties of $\Pi$. Before that, we will start by introducing some predicates which will be used in our definitions.

- $\{0, 1\} := \mathtt{isRegistered}(\mathtt{Id}, \sigma)$: This predicate takes as input the identity-password pair $(\mathtt{Id}, \mathtt{pwd})$, the current state of the system and outputs 1 if the user was previously registered with the corresponding identity $\mathtt{Id}$; otherwise, it outputs 0.
- $\{\mathtt{pwd}, \perp\} \leftarrow \mathtt{PullPwd}(\mathtt{Id}, \sigma)$: This algorithm takes as input the user identity $\mathtt{Id}$ and current state $\sigma$, and outputs $\mathtt{pwd}$ the associated password to $\mathtt{Id}$ if $\mathtt{isRegistered}(\mathtt{Id}, \sigma) = 1$; otherwise, it outputs $\perp$.

**Definition 10** (Correctness)**.** *Let* $\Pi = (\mathsf{RegisterNewUser}, \mathsf{Login})$ *be a password-based authentication mechanism,* $P$ *a binary predicate* [10]*, and* $\mathsf{EventType} = \{\mathsf{Register}, \mathsf{Login}\}$ *be the set of registration/login actions. Then, for all sequence of registration/login events* $U_1, \ldots, U_i \in \mathcal{ID} \times \mathcal{PWD} \times \mathsf{EventType}$ *resulting the state* $\sigma_i$*, the correctness of* $\Pi$ *enforces two following conditions:*

- ***Login Correctness:*** *For all events*

$$U_{i+1} = (\mathtt{Id}, \mathtt{pwd}, \mathsf{Login}) \in \mathcal{ID} \times \mathcal{PWD} \times \mathsf{EventType}$$

*we have* $(\sigma_{i+1}, \mathtt{acpt}) \leftarrow \mathsf{Login}(\sigma_i, \mathtt{Id}, \mathtt{pwd})$ *if*

$$i = \mathtt{isRegistered}(\mathtt{Id}, \mathtt{pwd}, \sigma_i)$$

*Intuitively, this basically means that the user with identity* $\mathtt{Id}$ *has already successfully registered with the corresponding password* $\mathtt{pwd}$ *and for all future login events using the the id-password pair* $(\mathtt{Id}, \mathtt{pwd})$ *the login algorithm* $\mathsf{Login}$ *always outputs* $\mathtt{acpt}$*.*

[10] The predicate determines if two passwords' distance satisfies the target distance metric or not. For example, if the distance metric is Hamming 2, for two password $\mathtt{pwd}_1, \mathtt{pwd}_2$, we have $P(\mathtt{pwd}_1, \mathtt{pwd}_2) = 1$ if $\mathtt{Ham}(\mathtt{pwd}_1, \mathtt{pwd}_2) \leq 2$

- $(P,\theta)$-***Typo Resilience:*** *Let $P$ be binary predicate, $0 \leq \theta \leq 1$ be an arbitrary real number, $U_{i+1} = (\mathtt{Id}_{i+1}, \mathtt{pwd}_{i+1}, \mathsf{Login}) \in \mathcal{ID} \times \mathcal{PWD} \times \mathsf{EventType}$ be a login event s.t. $1 := \mathtt{isRegistered}(\mathtt{Id}_{i+1}, \mathtt{pwd}_j, \sigma_i) \wedge P(\mathtt{pwd}_{i+1}, \mathtt{pwd}_j)$ for some $1 \leq j \leq i$, and $(\sigma_{i+1}, \mathtt{rjct}) \leftarrow \mathsf{Login}(\mathtt{Id}_{i+1}, \mathtt{pwd}_{i+1}, \sigma_i)$. We say $\Pi$ is $(P,\theta)$-Typo resistant if after login event $U_k = (\mathtt{Id}_{i+1}, \mathtt{pwd}_j, \mathsf{Login})$ for some $k > i+1$, for all $k' > k$ we have $(\sigma_{k'+1}, \mathtt{acpt}) \leftarrow \mathsf{Login}(\mathtt{Id}_{i+1}, \mathtt{pwd}_{i+1}, \sigma_{k'})$ with probability at least $\theta$. Intuitively, this implies that after an unsuccessful login with a password which has small distance to the original password, if we have a login with the original password, then, we have the chance of successful login (at least with probability $\theta$) with the miss-typed password and the that typo will be added to the cache of password which will grant successful login.*

# G Typo Vault: Security Definitions

In this section, we formally describe the security definitions based on a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. In the game we use a predicate which checks if the received login query is valid or not.

**Definition 11** (Valid Login Query)**.** *Let $\mathtt{pwd}_0, \mathtt{pwd}_1, \mathtt{pwd} \in \mathcal{PWD}$ be three passwords from password space $\mathcal{PWD}$. We say that the login query $(\mathtt{pwd}_0, \mathtt{pwd}_1, \mathtt{pwd})$ is a valid query under predicate $P$ if we either have*

*(1) $\mathtt{pwd}_0 = \mathtt{pwd}_1$ if $1 = P(\mathtt{pwd}_0, \mathtt{pwd})$ or $1 = P(\mathtt{pwd}_1, \mathtt{pwd})$.*

*(2) Or $P(\mathtt{pwd}_0, \mathtt{pwd}) = 0 \wedge P(\mathtt{pwd}_1, \mathtt{pwd}) = 0$.*

*We denote $\mathtt{ValidLginQuery}(\mathtt{pwd}_0, \mathtt{pwd}_1, \mathtt{pwd}) = 1$ when the query is valid with regard to the mentioned two conditions, otherwise we have $\mathtt{ValidLginQuery}(\mathtt{pwd}_0, \mathtt{pwd}_1, \mathtt{pwd}) = 0$.*

**Definition 12** (Typo Privacy)**.** *We say that a password-based authentication scheme $\Pi$ is $(t(\lambda), q(\lambda), \epsilon(t,q,\lambda))$-typo private under the predicate $P$ if for all adversaries $\mathcal{A}$ running in time $t$ and making at most $q$ queries to* Login/Register *we have*

$$\Pr\left[\mathtt{Experiment}_{\Pi,\mathsf{Typo-Privacy}}(\mathcal{A},\lambda,q) = 1\right] \leq \epsilon(t,q,\lambda) \tag{15}$$

In Experiment 16 we defined two main oracles that the adversary can make at most $q$ queries to them which are RegisterQuery and LoginQuery. RegisterQuery receives an ID and password, and assuming that it knows the current state of the system, first checks if the ID is registered previously or not. If not, it registers the user by running RegisterNewUser as described in Experiment 16 and updates the state. LoginQuery receives the and ID and a pair of two PWDs $\mathtt{pwd}_0$ and $\mathtt{pwd}_1$ as a request for login attempt. Similarly, assuming that oracle knows the current state of the system, the oracle uses $\mathtt{pwd}_b$ to run the login algorithm Login as described in Experiment 16. We also defined the time variable $t$ to track the number of queries and the state updates when the adversary access to the oracles.

We remark that there are other security definitions like *offline distinguishing*, *offline guessing* and *Random-or-Real* which are defined for a typo tolerable password-based authentication scheme and introduced and discussed in [CWP+17]. However, due to the page limitation we ignore to formally review and prove them. However, we should highlight that our construction also provide these security properties. Basically, the source of difference between our proposal and the TypTop mechanism is the underlying the public key encryption scheme. Moreover, it is worth mentioning that their Cheetarjee et al's construction dose not support the security definition we just introduced, i.e., Definition 12.

### G.0.1 No Typo Privacy for Original TypTop

In what fallows we will show that TypTop [CWP+17] does not satisfy Typo Privacy we described in Experiment 16. Before that, we just briefly review TypTop mechanism and then we will show why their proposal does not support our suggested security property.

**Experiment 16.**

$\underline{\mathtt{Experiment}_{\Pi,\mathsf{Typo-Privacy}}[\mathcal{A},\lambda,q]}$:

- **Init**
  (1) $\mathcal{C}$ runs $\sigma_0 \leftarrow \mathsf{Initialization}(1^\lambda)$.
  (2) $\mathcal{C}$ randomly selects $b \in_R \{0,1\}$.
  (3) $\mathcal{C}$ sets $t = 0$ as the starting time.
- **Query phase**: $\mathcal{A}$ makes at most $q$ queries to RegisterQuery/LoginQuery oracles:
  - $\mathsf{RegisterQuery}(\mathtt{Id}_i, \mathtt{pwd}_i)$
  - $\mathsf{LoginQuery}(\mathtt{Id}_i, \mathtt{pwd}_{i,0}, \mathtt{pwd}_{i,1})$
- **Guess**: Let $\mathtt{view} = (\lambda, \sigma_0, \sigma_1, d_1, \ldots, \sigma_q, d_q)$.
  - $b' = \mathcal{A}(\mathtt{view})$.
- **Experiment Output**:
  $\mathtt{Experiment}_{\Pi,\mathsf{Typo-Privacy}}[\mathcal{A},\lambda,q] = 1 \iff b' == b$

$\underline{\mathsf{RegisterQuery}(\mathtt{Id},\mathtt{pwd})}$:

(1) If $t > q$ return $\perp$. // at most $q$ queries is allowed.
(2) set $\sigma = \sigma_t$
(3) $(\sigma_{t+1}, d_{t+1}) \leftarrow \mathsf{RegisterNewUser}(\sigma, \mathtt{Id}, \mathtt{pwd})$
(4) set $t = t + 1$.

$\underline{\mathsf{LoginQuery}(\mathtt{Id},\mathtt{pwd}_0,\mathtt{pwd}_1)}$:

(1) If $t > q$ return $\perp$ // at most $q$ queries are allowed.
(2) set $\sigma = \sigma_t$
(3) if $1 = \mathtt{isRegistered}(\mathtt{Id}, \sigma)$
- $\mathtt{pwd}_{\mathtt{Id}} = \mathtt{PullPwd}(\mathtt{Id}, \sigma)$
- if $1 = \mathtt{ValidLginQuery}(\mathtt{pwd}_0, \mathtt{pwd}_1, \mathtt{pwd}_{\mathtt{Id}})$
  - $(\sigma_{t+1}, d_{t+1}) \leftarrow \mathsf{Login}(\mathtt{Id}, \mathtt{pwd}_b, \sigma)$.
  - set $t = t + 1$.
- else
  - set $d_{t+1} = \mathtt{rjct}$, $\sigma_{t+1} = \sigma$, $t = t + 1$

(4) else
- set $\sigma_{t+1} = \sigma, d_{t+1} = \mathtt{rjct}$
- set $t = t + 1$

Figure 4: Formal description "$\mathtt{Experiment}_{\Pi,\mathsf{Typo-Privacy}}$" which the experiment defining the typo privacy. The experiment is defined based on the interaction between unlimited adversary $\mathcal{A}$ and the challenger $\mathcal{C}$.

Let $\mathtt{pwd}$ be the user's password. In TypTop, the server uses a password-based key derivation function to extract the secret key $k = \mathtt{PKDF}(\mathtt{pwd})$ and also uses public key cryptography and assigns a pairs of secret-public key $(pk, sk)$ to each user in time of registration. The user assigned public key will be stored in the sever along with his/her other credentials. Moreover, let $\mathtt{AE} = (\mathtt{AE.Enc}, \mathtt{AE.Dec})$ be an authenticated encryption scheme. Then, the server encrypts $sk$ using $k$ to store the ciphertext $C_{sk} = \mathtt{AE.Enc}_k(sk)$ in the server side. After an unsuccessful login attempt using a miss typed password $\mathtt{pwd}'$, the server will encrypt the typo $\mathtt{pwd}'$ using $pk$, and adds it to the waiting list. In the future login using the original password $\mathtt{pwd}$, the server extracts $k = \mathtt{PKDF}(\mathtt{pwd})$, and decrypts $C_{sk}$ to extract the secret key $sk$. Then the server uses the secret key $sk$ to decrypt all the ciphertexts appeared in the waiting list. Finally, those that have small edit distance to the original password $\mathtt{pwd}$ have the chance to be added to the cache of valid passwords/typos which will grant successful login for the future attempts.

Looking at the adversary's ability/view, we can see that the designed TypTop dose not satisfy Typo-privacy. Based on the experiment description, we can see that the adversary knows the passwords and therefore she can decrypt the challenge ciphertext to determine which password is chosen by the challenger. This was a high level intuition of the security issue and in what follows you may find more formal technical details of the mentioned issue. For simplicity we assume that number of queries is at most $q = 2$ the first one is the registration query of a user with identity $\mathtt{Id}$ and password $\mathtt{pwd}$. And the second query is t a login query with password pairs $\mathtt{pwd}_0, \mathtt{pwd}_1$ of the adversary choice and we have $P(\mathtt{pwd}_0, \mathtt{pwd}) \neq 1$ (so $\mathtt{pwd}_0$ and $\mathtt{pwd}_1$ are not necessarily equal while based on the condition of the experiment if $P(\mathtt{pwd}, \mathtt{pwd}_0) = 1$, then they have to be equal i.e., $\mathtt{pwd}_0 = \mathtt{pwd}_1$, i.e., $\mathtt{ValidLginQuery}(\mathtt{pwd}_0, \mathtt{pwd}_1, \mathtt{pwd}) = 1$). Regarding the description of TypTop, the challenger will add the ciphertext $C_{\mathtt{pwd}_b} = \Pi_{\mathsf{PubKey}}.\mathsf{Enc}(pk, \mathtt{pwd}_b)$ to the updated state as the encryption of a potential typo. Based on the description of the game, the adversary chooses the registration password $\mathtt{pwd}$ and knows. So simply can extract the authenticated decryption key $k = \mathtt{PKDF}(\mathtt{pwd})$ and computes the secret key $sk = \Pi_{\mathsf{PubKey}}.\mathsf{Dec}(k, C_{sk})$. Finally, the adversary computes $\mathtt{pwd}_b = \Pi_{\mathsf{PubKey}}.\mathsf{Dec}(sk, C_{\mathtt{pwd}_b})$ and simply outputs $b$ ($\mathcal{A}$ knows both $\mathtt{pwd}_0, \mathtt{pwd}_1$).

The above attack represents the simplified version of the actual Typo privacy and actually is a weaker definition. However, as the TypTop scheme does not provide this weaker requirement, it the does not satisfy Typo privacy as well.

# H Personalized Typo Tolerance Mechanism from Conditional Encryption

In this part we will describe our generic construction of a password typo vaults which safely caching incorrect login attempts with conditional encryption. In the rest of the paper we call it " CondTypTop" to imply construction of TypTop which is designed using conditional encryption. The main building blocks of our proposed construction are conditional encryption $\Pi_{\mathtt{CE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$, authenticated encryption $\Pi_{\mathtt{AE}} = (\mathsf{Enc}, \mathsf{Dec})$ and password-based key derivation function $\mathtt{PKDF}$[11]. Let the underlying conditional encryption be over the binary predicate $P$. Considering the mentioned building blocks, our construction of $\Pi_P = (\mathsf{Initialization}, \mathsf{RegisterNewUser}, \mathsf{Login})$ is described in Figure 6.

Intuitively, we should highlight that construction is similar to TypTop, however the main and basic idea and difference is replacing the traditional public key encryption part with our proposed conditional encryption scheme. So after an incorrect attempt using the typo included password $\mathtt{pwd}'$ we encrypt it using conditional encryption using the encryption of original password ciphertext. In fact, in this case $\mathtt{pwd}'$ will be considered as the payload of $\Pi_{\mathtt{CE}}.\mathsf{CEnc}$. For this aim, we have to encrypt the original password $\mathtt{pwd}$ using $C_{\mathtt{pwd}} = \Pi_{\mathtt{CE}}.\mathsf{Enc}_{pk}(\mathtt{pwd}; r)$ and store it in the server along with user's public key $pk$. Then, in future login with the miss typed password $\mathtt{pwd}'$, we add the ciphertext $C_{pwd'} = \Pi_{\mathtt{CE}}.\mathsf{CEnc}_{pk}(C_{\mathtt{pwd}}, \mathtt{pwd}', \mathtt{pwd}'; r')$ to our waiting list. So, in the future login with the original password $\mathtt{pwd}$ we extract $k_{\mathtt{pwd}} = \mathtt{PKDF}(\mathtt{pwd})$ and decrypt $sk = \Pi_{\mathtt{AE}}.\mathsf{Dec}(k_{\mathtt{pwd}}, C_{sk})$. Finally, if $P(\mathtt{pwd}, \mathtt{pwd}') = 1$, the conditional decryption returns the underlying

[11] which takes as input the password and deterministically extracts a key for symmetric key encryption scheme and we have $k_{\mathtt{pwd}} = \mathtt{PKDF}(\mathtt{pwd})$

payload (here is the password with small typo $\mathtt{pwd}'$) $\mathtt{pwd}' = \Pi_{\mathtt{CE}}.\mathsf{Dec}_{sk}(C_{\mathtt{pwd}'})$; otherwise, the decryption algorithm returns uniformly random element chosen from the password space $\mathcal{PWD}$.

In what follows, we will provide the detailed description of algorithms Initialization, RegisterNewUser and Login.

- $\sigma_0 \leftarrow \mathsf{Initialization}(1^\lambda)$: This algorithm takes as input the security parameter $\lambda$, and sets the initial state $\sigma_0 = (\lambda, S_W, S_T, W, T)$ in which $W$ is allocated waiting list, $T$ the cache of valid typos granting successful login. $S_W$ and $S_T$ are respectively the size of waiting list and cache size allocated for each user. Initially, as we don't have any registered user, $W$ and $T$ are empty.

- $(\sigma', d = \{\mathtt{acpt}, \mathtt{rjct}\}) \leftarrow \mathsf{RegisterNewUser}(\sigma, \mathtt{Id}, \mathtt{pwd})$: Let $\sigma$ be the current state of the system, and $(\mathtt{Id}, \mathtt{pwd})$ be the pair of password and user identity. If $1 = \mathtt{isRegistered}(\sigma, \mathtt{pwd})$, the algorithm outputs $(\sigma, d = \mathtt{rjct})$; otherwise, the algorithm computes symmetric key $k_{\mathtt{pwd}} = \mathtt{PKDF}(\mathtt{pwd}||\mathtt{salt}_{\mathtt{Id}})$ in which $\mathtt{salt}_{\mathtt{Id}} \in_R \{0,1\}^\lambda$ is the assigned user's slat, and $(pk_{\mathtt{Id}}, sk_{\mathtt{Id}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Then, algorithm computes $C_{\mathtt{Id},\mathtt{pwd}} = \Pi_{\mathtt{CE}}.\mathsf{Enc}_{pk_{\mathtt{Id}}}(\mathtt{pwd}; r)$ under random coin $r \in_R \{0,1\}^\lambda$. Then the algorithm will assign a waiting list $W_{\mathtt{Id}}$ of size $S_W = \sigma[1]$ and will fill $W_{\mathtt{Id}}$ with conditional encryption of garbage messages, i.e., $W_{\mathtt{Id}} = \Pi_{\mathtt{CE}}.\mathsf{CEnc}_{pk_{\mathtt{Id}}}(C_{\mathtt{Id},\mathtt{pwd}}, r', r'; r")$ s.t. $r' \in_R \mathcal{PWD}$[12] and $r'' \in_R \{0,1\}^\lambda$. Finally, the array $T_{\mathtt{Id}}$ of size $S_T = \sigma[2]$ as a cache of passwords which granting successful logins will be assigned to the user with identity $\mathtt{Id}$ such that $T_{\mathtt{Id}}[0] = \Pi_{\mathtt{AE}}.\mathsf{Enc}(k_{\mathtt{pwd}}, sk_{\mathtt{Id}})$ . To track the number of logins, the algorithm sets $c_{\mathtt{Id}}$ as the counter which will be increased by one after each attempt for login. We set $\sigma_{\mathtt{Id}} = (\mathtt{Id}, \mathtt{salt}_{\mathtt{Id}}, pk_{\mathtt{Id}}, W_{\mathtt{Id}}, T_{\mathtt{Id}}, C_{\mathtt{Id},\mathtt{pwd}}, c_{\mathtt{Id}})$ The updated state will be $\sigma' = (\sigma, \sigma_{\mathtt{Id}})$ and the algorithm outputs $(\sigma', d = \mathtt{acpt})$.

- $(\sigma', d = \{\mathtt{acpt}, \mathtt{rjct}\}) \leftarrow \mathsf{Login}(\sigma, \mathtt{Id}, \mathtt{pwd})$: The algorithm first checks if the user is registered and output $(\sigma' = \sigma, d = \mathtt{rjct})$ if $0 = \mathtt{isRegistered}(\sigma, \mathtt{Id})$; otherwise, it extracts the state $\sigma_{\mathtt{Id}} \in \sigma$ which was previously assigned to the user $\mathtt{Id}$. Then it obtains the salt $\mathtt{salt}_{\mathtt{Id}} = \sigma_{\mathtt{Id}}[1]$ and computes $k_{\mathtt{pwd}} = \mathtt{PKDF}(\mathtt{pwd}||\mathtt{salt}_{\mathtt{Id}})$. Then considering the cache $T_{\mathtt{Id}} = \sigma_{\mathtt{Id}}[4]$, the algorithm search for the ciphertext $T_{\mathtt{Id}}[i]$ for all $1 \le i \le S_T$ such that $(1, sk_{\mathtt{Id}}) = \Pi_{\mathtt{AE}}.\mathsf{Dec}(k_{\mathtt{pwd}}, T_{\mathtt{Id}}[i])$, this is a successful login and the algorithm sets $d = \mathtt{acpt}$; otherwise, $d = \mathtt{rjct}$. Now the counter is updated i.e., $\sigma_{\mathtt{Id}}[6] = \sigma_{\mathtt{Id}}[6] + 1$, we have two cases:

  (1) **Successful login ($d = \mathtt{acpt}$)**: Now the algorithm will process waiting list using the extracted secret key $sk_{\mathtt{Id}}$. For this aim, the algorithm will decrypt all the ciphertexts in $W_{\mathtt{Id}} = \sigma_{\mathtt{Id}}[3]$ as $m_j = \Pi_{\mathtt{CE}}.\mathsf{Dec}_{sk_{\mathtt{Id}}}(W_{\mathtt{Id}}[j])$ for all $1 \le j \le S_W$. For those $m_j$ that have $P(\mathtt{pwd}, m_i) = 1$ we add them to the cache $T_{\mathtt{Id}}$ if we have empty space for new typos in $T_{\mathtt{Id}}$. We can assign a weight to $m_i$ based on its appearance in the sequence $m_1, \ldots, m_{S_W}$ and randomly selects them based on their weight until the cache become full. For the selected $m_k$ the algorithm will use $k_{\mathtt{pwd},m_k} = \mathtt{PKDF}(m_k||\mathtt{salt})$ to compute

  $$C_{\mathtt{pwd},m_k} = \Pi_{\mathtt{AE}}.\mathsf{Enc}_{k_{\mathtt{pwd},m_k}}(sk_{\mathtt{Id}}) \ .$$

  Now, a random shuffling will be applied on the elements of cache with the constrain that the first element is always associated to the original password. After processing waiting list, similar to the RegisterNewUser, the algorithm fill the waiting list with garbage ciphertexts and updates $W_{\mathtt{Id}}$. In this, we should highlight that the updated $\sigma'$ is the same as $\sigma$ and the only different is the updated cache/waiting list $T_{\mathtt{Id}}/W_{\mathtt{Id}}$ and. This is the successful login and the algorithm outputs and the output is $\sigma' = \sigma$

  (2) **Unsuccessful login ($d = \mathtt{rjct}$)**: In this case we need to conditionally encrypt $\mathtt{pwd}$ and add it to the waiting list. So, first we extract $C_{\mathtt{Id},\mathtt{pwd}'} = \sigma_{\mathtt{Id}}[5]$ and compute

  $$C_{\mathtt{pwd},\mathtt{pwd}'} \leftarrow \Pi_{\mathtt{CE}_{pk_{\mathtt{Id}}}}.\mathsf{CEnc}_{pk_{\mathtt{Id}}}(C_{\mathtt{Id},\mathtt{pwd}'}, \mathtt{pwd}, \mathtt{pwd}; r)$$

  in which $r \in_R \{0,1\}^\lambda$ (remind that $\lambda = \sigma[0]$). Assuming the updated value of counter $c_{\mathtt{Id}} = \sigma_{\mathtt{Id}}[6]$, the algorithm computes $i = c_{\mathtt{Id}} \mod S_W$ and replace the $i$-th element of the waiting list. Then it

[12] We remind you that $\mathcal{PWD}$ is the space of all possible passwords.

will randomly shuffle the waiting list and update the state as $\sigma'$. We highlight that, $\sigma'$ is different with $\sigma$ related to the counter and the changes applied to the waiting list.

Now the login attempt is done and the algorithm outputs $(\sigma', d)$.

## H.1 Security proof of CondTypTop

In this part we will prove the security of the CondTypTop. We should highlight that we just concentrate on the typo-privacy that we defined in this paper. The other security properties defined for TypTop are still preserving as our construction is similar to the original TypTop. So, due to page limitation, we leave discussing them here and with similar reasoning all the remaining security properties can be proved in the same way.

To prove that CondTypTop provides Typo-privacy, we use hybrid argument and will define three hybrids. Then, using the security of conditional encryption, we show that all these hybrids are computational indistinguishable. Finally, by indistinguishablity of the first and the last hybrid, we conclude that the our CondTypTop has typo privacy. Intuitively, the first hybrid is the original typo privacy game. In the second hybrid, we take the current state of the system and replace all the ciphertexts that are the output of the $\mathsf{CEnc}$ algorithm with the output of $\mathtt{Sim}(pk)$ of conditional encryption. As we assumed that the underlying conditional encryption is secure, inherently such simulator exists. Based on the construction of CondTypTop, the ciphertexts we replacing are the conditional encryption of $\mathtt{pwd}_{i,b}$ for all $i$ such that the $i$-th query is $\mathsf{LoginQuery}$ query. Finally, in the last hybrid, we just replace $\mathtt{Sim}(pk)$ with conditional encryption of $\mathtt{pwd}_{i,1-b}$. In what follows we formally prove the typo privacy of CondTypTop.

**Theorem 17.** *Given the $(t, q, \epsilon)$-secure conditional encryption $\Pi_{\mathtt{CE}} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{CEnc}, \mathsf{Dec})$, then CondTyoTop $\Pi$ descried in **Construction 20** is $(t', q', \epsilon')$-typo private.*

**Proof of Theorem 17:** To prove the security of our CondTyoTop, we should highlight that for the ciphertextsof conditional encryption which presented in the final state, two main cased can be considered. Without loss of generality, let the $i$-th query be a $\mathsf{LoginQuery}$ query. Then state of the system contains the conditional encryption $C_{ib} = \mathsf{CEnc}_{pk}(C_{\mathtt{pwd}_{\mathtt{Id}}}, \mathtt{pwd}_{i,b})$ in which $C_{\mathtt{pwd}_{\mathtt{Id}}} = \mathsf{Enc}_{pk}(\mathtt{pwd}_{\mathtt{Id}})$ for some previously registered user with identity $\mathtt{Id}$. So the two cases are as follows:

- $\mathtt{pwd}_{i,0} = \mathtt{pwd}_{i,1}$. In this case, both ciphertexts are statistically indistinguishable and their input plaintexts are exactly the same. So no adversary can distinguish between their ciphertexts.
- $P(\mathtt{pwd}_{i,0}, \mathtt{pwd}_{\mathtt{Id}}) = 0 \wedge P(\mathtt{pwd}_{i,0}, \mathtt{pwd}_{\mathtt{Id}}) = 0$. For this case we need to prove adversary's advantage to win in the experiment is polynomially close to the advantage of adversary who breaks the security of conditional encryption. We will show it by defining three following hybrids **Hybrid 0**, **Hybrid 1** and **Hybrid 2**.

As we mentioned before, to prove the security of **Construction 20**, we defined three hybrids and then prove all these hybrids are indistinguishable. Indistinguishability of these hybrids implies that the advantage of the adversary to win in $\mathtt{Experiment}_{\Pi,\mathsf{Typo-Privacy}}$ is negligible.

**Hybrid 0**: This is the original experiment $\mathtt{Experiment}_{\Pi,\mathsf{Typo-Privacy}}$.

**Hybrid 1**: Let $\sigma$ be the final state of the typo privacy experiment defined in **Hybrid 0**. Suppose $Q = \{C_{i,b} \in \sigma\}$ be set of all conditional encryption of $\mathtt{pwd}_{i,b}$ for all the login queries $1 \leq i \leq q'$. Due to the security of conditional encryption, we know that there exists simulator $\mathtt{Sim}(pk)$ who simulates the ciphertext of conditional encryption scheme. In this hybrid, we replace all $C_{i,b} \in Q$ with $C'_{i,b} \leftarrow \mathtt{Sim}(pk)$.

**Hybrid 2**: In this hybrid we simply replace all $C'_{i,b}$ with

$$C_{i,1-b} = \mathsf{CEnc}_{pk}(C_{\mathtt{pwd}_i d}, \mathtt{pwd}_{i,1-b})$$

.

### H.1.1 Indistinguishability of the hybrids

. Due to the security of conditional encryption, the advantage of adversary $\mathcal{A}$ to distinguish between **Hybrid 0** and **Hybrid 1** is $\epsilon$. More formally, we should highlight that as $P(\mathtt{pwd}_{i,b}, \mathtt{pwd}_{\mathtt{Id}}) = 0$, the simulator can simply select a random number and encrypt it using the public key $pk$. As the security of conditional encryption guarantees that if $P(m_1, m_2) = 1$, then the resulting ciphertext will be the encryption of a message chosen uniformly at random. And this is basically what our simulator $\mathtt{Sim}(pk)$ does. With similar argument, the adversary advantage in distinguishing between the hybrids **Hybrid 1** and **Hybrid 2** is also $\epsilon$. As a result, the advantage of adversary $\mathcal{A}$ spending time $t' = t$ to distinguish between **Hybrid 0** and **Hybrid 1** is $\epsilon' = \epsilon$.

□

# I Constructions

**Construction 18.**

$\underline{(pk, sk) = \mathsf{KeyGen}(1^\lambda)}$:

(1) Pick random $r \in_R \{0,1\}^\lambda$

(2) Compute $(sk = (\beta, \mu), pk = (N, g)) \leftarrow \mathtt{P.KeyGen}(1^\lambda; r)$[a]

(3) If $|\Sigma|^{n+1} \geq \min(p, q)$ or $\mathtt{gcd}(N, (p-1)(q-1)) \neq 1$, repeat step (1); else, output $(sk, pk)$

$\underline{c_{m_1} \leftarrow \mathsf{Enc}_{pk}(m_1)}$:

(1) Pick random $r \in_R \{0,1\}^\lambda$

(2) Compute $\hat{m}_1 = \mathsf{ToInt}(m)$.

(3) Return $(0, c_{m_1} := \mathtt{P.Enc}(\hat{m}_1; r))$

$\underline{\tilde{c} = \mathsf{CEnc}_{pk}(c_{m_1}, m_2, m_3)}$:

(1) Parse $(b, c) \leftarrow c_{m_1}$ and $(N, g) \leftarrow pk$.

(2) If $b = 1$ or $\mathtt{gcd}(c, N) \neq 1$ return $\perp$.

(3) Compute $\hat{m}_3 = \mathsf{ToInt}(m_3)$, $\hat{m}_2 = \mathsf{ToInt}(m_2)$ and select $R \in_R \mathbb{Z}_N$, $r_2 \in_R \mathbb{Z}_N^*$

(4) $c^{=} = g^{\hat{m_3}} \cdot [(R \boxtimes c) \boxplus \mathtt{P.Enc}_{pk}((-R) \cdot m_2; r_2)]$

(5) Return $\tilde{c} = (1, c^{=})$

$\underline{m := \mathsf{Dec}_{sk}(\tilde{c})}$:

(1) Parse $(b, c) \leftarrow \tilde{c}$

(2) Compute $x \leftarrow \mathtt{P.Dec}_{sk}(c)$.

(3) compute $m \leftarrow \mathsf{ToInt}^{-1}(x)$

(4) If $b = 1$ and $m > |\Sigma|^{n+1}$ return $\perp$

(5) Otherwise return $m$

[a] We use the simple variant of Pallier key generation which picks primes $p$ and $q$ of equivalent length and sets $N = pq$, $g = N + 1$, $\beta = \mathsf{lcm}(p-1, q-1)$ and $\mu = \varphi(N)^{-1} \mod N$ where $\varphi(N) = (p-1)(q-1)$. See details in Appendix A.

**Construction 19.**

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

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

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

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` $\hat{m}[i] = \mathsf{ToInt}(m[i]), c_i = \mathtt{P}.\mathsf{Enc}_{pk}(\hat{m}[i]; r_i)$
3. `output` $c = (b = 0, c_1, \ldots, c_n)$.

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

1. `parse` $m' = (m'[1], \ldots, m'[n]) \in \Sigma^n$, $r = (r_1, \ldots, r_n) \in (\mathbb{Z}_N^*)^n$, $c_m = (b, c_1, \cdots, c_n)$
2. `compute` $\hat{m}'[i] = \mathsf{ToInt}(m'[i])$ to map each letter to integer.
3. `if` $b = 1$, `output` $\perp$.
4. $\forall 1 \leq i \leq n$, `compute` $\tilde{c}'_i = \mathtt{P}.\mathsf{Enc}_{pk}(\hat{m}'[i]; r_i) \boxminus c_i$.
5. Pick $K \in_R \{0,1\}^\lambda$, compute $c_{AE} = \mathsf{Auth.Enc}_K(m'')$.
6. $((i_1, [\![s]\!]_1), \ldots, (i_n, [\![s]\!]_n)) \leftarrow \mathtt{SS}.\mathsf{ShareGen}(n, n-l, K)$.
7. $\forall 1 \leq i \leq n$ : `sample` $a_i \in_R \left[1, \left\lfloor \frac{N-1}{2^\lambda} - 1 \right\rfloor\right]$, $\tilde{r}_i \in_R \mathbb{Z}_N^*$
8. $\forall 1 \leq i \leq n$ : `compute` $y_i = [\![s]\!]_i + a_i 2^\lambda, \tilde{c}_i = \mathtt{P}.\mathsf{Enc}_{pk}(y_i; \tilde{r}_i) \boxplus (R_i \boxtimes \tilde{c}'_i)$
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$(\cdot)$

- `parse` $\tilde{c} = (\tilde{c}_1, \cdots, \tilde{c}_n, c_{AE})$, `AND set` $m = \perp$
- `for` $1 \leq i \leq n$ : $\hat{y}_i = \mathtt{P}.\mathsf{Dec}_{sk}(\tilde{c}_i)$, $[\![s']\!]_i = \mathsf{RDec}(\hat{y}_i) = \hat{y}_i \mod 2^\lambda$.
- For all $\binom{n}{n-\ell}$ possible $S \subseteq [n]$ with $|S| = n - \ell$,
  - `compute` $K_S = \mathsf{SecretRecover}\left(\{(i, [\![s']\!]_i)\}_{i \in S}\right)$
  - $(v', \hat{m}'') = \mathsf{Auth.Dec}_{K_S}(c_{AE})$, `if` $v' = 1$, `return` $m = \hat{m}''$[a]

2.2 `if` $b = 0$ // The ciphertext is extracted from Enc$(\cdot)$

- `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` $m = \mathsf{ToInt}^{-1}(\hat{m}_1) \| \ldots \| \mathsf{ToInt}^{-1}(\hat{m}_n)$

[a] Here $m = \perp$ as for all possible $K_S$ the $\mathsf{AuthDec}_{K_S}$ algorithm failed.

Figure 5: Concrete construction of conditional encryption over binary predicate $P_{\ell,\mathtt{Ham}}$ using secret sharing.

**Construction 20.**

$\sigma_0 \leftarrow \mathsf{Initialization}(1^\lambda)$:
Given the security parameter $\lambda$ as input, this algorithm acts as follows.

1. For all $1 \leq i \leq L$ sample $w_i \in_R \mathcal{PWD}$ uniformly at random, and Set $W = [\Pi_{\mathtt{Cond}}.\mathsf{Enc}(w_1), \ldots, \Pi_{\mathtt{Cond}}.\mathsf{Enc}(w_L)]$.
2. Set $T = []$ as the list for collecting legitimate typos
3. Set $\sigma_0 = (W, T)$

$(\sigma_i, d = \{\mathtt{acpt}, \mathtt{rjct}\}) \leftarrow \mathsf{RegisterNewUser}(\sigma_{i-1}, \mathtt{Id}, \mathtt{pwd})$:

1. For all $1 \leq i \leq L$ sample $w_i \in_R \mathcal{C}_{\Pi_{\mathtt{Cond}}}$ [a] , and Set $W = [w_1, \ldots, w_L]$, and set $W_{\mathtt{Id}} = W$.
2. Set $T_{\mathtt{Id}} = []$ //As the list for collecting legitimate typos
3. Compute $b = \mathsf{IsRegistered}(\sigma_{i-1}, \mathtt{Id})$.

   If $b = 0$, set $d = \mathtt{acpt}$ and go to step 4;
   If $b = 1$, set $d = \mathtt{rjct}$, break and return $(\sigma_i, d)$
4. Run $(pk_{\mathtt{Id}}, sk_{\mathtt{Id}}) \leftarrow \Pi_{\mathtt{Cond}}.\mathsf{KeyGen}(1^\lambda)$.
5. Set $k_{\mathtt{Id},\mathtt{pwd}} = PKDF(\mathtt{pwd})$, compute $C_{\mathtt{Id},\mathtt{pwd}} = AE.\mathsf{Enc}_{k_{\mathtt{Id},\mathtt{pwd}}}(sk)$, and $c_{\mathtt{pwd}} = \Pi_{\mathtt{Cond}}.\mathsf{Enc}_{pk_{\mathtt{Id}}}(\mathtt{pwd})$
6. Set $T_{\mathtt{Id}}[0] = C_{\mathtt{Id},\mathtt{pwd}}$, and $W_{\mathtt{Id}} = W$.
7. Update $\sigma_i = \mathtt{Append}(\sigma_{i-1}, (c_{\mathtt{pwd}}, pk_{\mathtt{Id}}, W_{\mathtt{Id}}, T_{\mathtt{Id}}))$.
8. `return` $(\sigma_i, d)$

$(\sigma_i, d = \{\mathtt{acpt}, \mathtt{rjct}\}) \leftarrow \mathsf{Login}(\sigma_{i-1}, \mathtt{Id}, \mathtt{pwd}')$:

1. Compute $s_{\mathtt{Id}} = \mathsf{Extract}(\mathtt{Id}, \sigma_{i-1})$.

   If $s_{\mathtt{Id}} = \perp$, return $(\sigma_{i+1} = \sigma_i, d = \mathtt{rjct})$ and break;
   Else, parse $s_{\mathtt{Id}} = c_{\mathtt{pwd}}, pk_{\mathtt{Id}}, W_{\mathtt{Id}}, T_{\mathtt{Id}}$
2. Compute $k = PKDF(\mathtt{pwd})$
3. If $\forall x \in T_{\mathtt{Id}}$, $\perp = AE.\mathsf{Dec}_k(x)$

   2.1 Set $d = \mathtt{rjct}$, compute $c_{\mathtt{pwd}'} = \Pi_{\mathtt{Cond}}.\mathsf{CEnc}(c_{\mathtt{pwd}}, \mathtt{pwd}')$, update $W_{\mathtt{Id}} = \mathtt{Append}(W_{\mathtt{Id}}, c_{\mathtt{pwd}'})$. //Adding typo's ciphertext to waiting list $W_{\mathtt{Id}}$.
   2.2 Set $s_{\mathtt{Id}} = (c_{\mathtt{pwd}}, pk_{\mathtt{Id}}, W_{\mathtt{Id}}, T_{\mathtt{Id}})$, compute $\sigma_i = \mathsf{Replace}(\mathtt{Id}, s_{\mathtt{Id}}, \sigma_{i-1})$, return $(\sigma_i, d)$, and break.
3. Find $x \in T_{\mathtt{Id}}$ s.t. $\perp \neq sk_{\mathtt{Id}} = AE.\mathsf{Dec}_k(x)$, and $\forall i \in [|W_{\mathtt{Id}}|]$ compute $pw_i = \Pi_{\mathtt{Cond}}.\mathsf{Dec}_{sk_{\mathtt{Id}}}(W_{\mathtt{Id}}[i])$,
   - if $1 = P(pw_i, \mathtt{pwd})$: compute $k_{\mathtt{Id},pw_i} = PKDF(pw_i)$, $C_{\mathtt{Id},pw_i} = AE.\mathsf{Enc}_{k_{\mathtt{Id},pw}}(sk_{\mathtt{Id}})$, and update $T_{\mathtt{Id}} = \mathtt{Append}(T_{\mathtt{Id}}, C_{\mathtt{Id},pw})$.
4. For all $1 \leq i \leq L$ sample $w_i \in_R \mathcal{C}_{\Pi_{\mathtt{Cond}}}$, and set $W = [w_1, \ldots, w_L]$, and set $W_{\mathtt{Id}} = W$.
5. Set $s_{\mathtt{Id}} = (c_{\mathtt{pwd}}, pk_{\mathtt{Id}}, W_{\mathtt{Id}}, T_{\mathtt{Id}})$, and compute $\sigma_i = \mathsf{Replace}(\mathtt{Id}, \sigma_i, s_{\mathtt{Id}})$.
6. Return $(\sigma_i, d = \mathtt{acpt})$

[a] We note that $w_i$s are random element sampled from the ciphertext space $\mathcal{C}_{\Pi_{\mathtt{Cond}}}$ associated to $\Pi_{\mathtt{Cond}}$.

Figure 6: Proposed generic construction of our password typo vaults for secure caching incorrect login attempts with conditional encryption