# hPRESS: A Hardware-enhanced Proxy Re-encryption Scheme using Secure Enclave

Fan Zhang, *Member, IEEE*, Ziyuan Liang, Cong Zuo,
Jun Shao, Jianting Ning, Jun Sun, Joseph K. Liu, and Yibao Bao

***Abstract*—Proxy re-encryption (PRE) allows a proxy to transform one ciphertext to another under different encryption keys while keeping the underlying plaintext secret. Because of the ciphertext transformability of PRE, there are many potential private communicating applications of this feature. However, existing PRE schemes are not as full-fledged as expected. The lack of necessary features makes them hard to apply in real-world scenarios. So far, there does not exist a unidirectional multi-hop PRE scheme with constant decryption efficiency and constant ciphertext size without extensions. Impractical performance and weak scalability also hinder PRE from most real-world applications. In this work, we present a new PRE scheme with secure hardware enclave named `hPRESS` (hardware-enhanced proxy re-encryption scheme using secure enclave). To the best of our knowledge, `hPRESS` is the first unidirectional multi-hop PRE scheme which achieves both constant decryption efficiency and constant ciphertext size without extensions. A detailed security analysis demonstrates that our proposal is CCA secure based on the security of the underlying encryption schemes and the secure enclave. We also implement a prototype based on Intel SGX, one of the most popular secure enclave techniques in recent years, and evaluate its performance. The experimental results show that, compared with previous PRE schemes, our `hPRESS` is almost one order of magnitude faster in terms of the decryption and transformation.**

***Index Terms*—Proxy Re-encryption, Secure Enclave, Trusted Computing, Intel SGX**

## I. Introduction

PROXY re-encryption (PRE) is a powerful cryptographic primitive, which allows a semi-trusted proxy to transform a ciphertext of user A (called delegator) to another ciphertext of user B (called delegatee). The two plaintexts of the original and transformed ciphertexts are the same, and they are never revealed (even to the proxy) during the transformation. If the transformed ciphertext can be further transformed, we call the underlying scheme as multi-hop PRE; otherwise, single-hop PRE. Moreover, if the transformation can happen in two directions between the delegator and the delegatee, we call the underlying scheme as bidirectional PRE; if it can only happen from the delegator to the delegatee, we call it as unidirectional PRE.

### A. Motivation

Most existing PRE schemes are far from real-world applications, which is our main motivation to propose a new practical PRE scheme. To illustrate the deficiencies of current PRE schemes, we consider several potential use cases of PRE, analyze features in need of a practical PRE scheme.

Digital right management (DRM) provides licensing restrictions to digital content for authenticated uses. It takes advantage of the online content distribution while preventing illegal redistribution of the content. Taban et al. [1] first proposed a secure and interoperable DRM architecture with PRE and re-signature, which is similar with PRE. They introduced an intermediate module called the Domain Interoperability Manager (DIM) to deal with the problem of content and license translation across different DRM regimes. DIM operators are regarded as semi-trusted proxies in their design. DIM converts a ciphertext computed under one key to another, without the DIM learning any information about the plaintext message or the secret keys of the content providers or other authenticated parties. Although PRE primitives perfectly fit in the scenario of DRM and DIM, there is still no DRM systems based on PRE in real-world applications. The reason is that existing PRE schemes do not provide all features demanded by a practical DRM system. First, DRM architecture needs a unidirectional PRE scheme. The content providers only allow DIM to re-encrypt from higher-level authenticated parties to lower-level local groups or individual devices, and forbid the inverse transformation. Content providers should not need to trust authenticated users in order for them to be able to forward the license back. The PRE scheme adopted by DRM also needs to be a multi-hop one. In most cases, the digital contents are not directly authenticated from the content providers to the final user devices. There are several proxy nodes in the authentication chain, and the contents are always authenticated from high-level parties to low-level ones. Thus the PRE scheme for a practical DRM system is a unidirectional multi-hop one, and it is also expected to support constant ciphertext

Fan Zhang is with College of Computer Science and Technology, Zhejiang University, Hangzhou, China. He is also with State Key Laboratory of Cryptology, Alibaba-Zhejiang University Joint Research Institution of Frontier Technolgies, and Zhejiang Lab. (Email: fanzhang@zju.edu.cn). Ziyuan Liang is with College of Information Science & Electronic Engineering, Zhejiang University, Hangzhou, China. (Email: liangziyuan@zju.edu.cn)

Cong Zuo and Joseph Liu are with Faculty of Information Technology, Monash University, Australia. Cong Zuo is also with Data61, CSIRO, Melbourne, Australia.

Jun Shao is with School of Computer and Information Engineering, Zhejiang Gongshang University, China.

Jianting Ning is with College of Mathematics and Informatics, Fujian Normal University, Fuzhou, China, and the School of Information Systems, Singapore Management University, Singapore. He is the corresponding author. (Email: jtning88@gmail.com)

Jun Sun is with School of Information Systems, Singapore Management University, Singapore.

Yibao Bao is with Security Department of Alibaba Group, Hangzhou, China.

size without extensions to keep a constant communication cost. It needs to support ciphertext transformation between arbitrary encryption schemes to support existing DRM applications. For the sake of performance, it is better for the PRE scheme to keep a constant decryption efficiency. However, there still does not exist PRE schemes covering all these features, which might block PRE away from constructing a DRM system in real-world applications.

There are many other potential applications, for instance, online car rental. However, most of these applications suffer from the same problem of DRM, thus a unidirectional, multi-hop scheme with better performance is needed. Since PRE was proposed, many cryptographers have made great efforts to develop better PRE schemes. However, cryptanalysts focus more on the correctness, the security, and functional features of PRE like unidirectionality or multi-hop constructions. Meanwhile, practical features are overlooked by PRE designers and cryptanalysts, such as scalability, computational and communication costs, which directly determine whether PRE can be adopted by real-world applications or not. Constant decryption efficiency and constant ciphertext size help keep the computational and communication cost in an acceptable and practical range. In addition, the ability of supporting arbitrary ciphertext can increase the scalability of the schemes significantly.

In view of these practical problem, we aim to find a practical PRE scheme for real-world applications. However, designing and implementing a practical PRE scheme is highly non-trivial. It took about two decades to propose the first unidirectional multi-hop PRE scheme with CCA security [2] since PRE was first proposed in 1998. It is getting more and more challenging to achieve practical features while retaining these security and functional features. We propose a PRE scheme based on secure enclave to address the issues of existing PRE, as it is hard to combine them into a single scheme with pure cryptography. What makes our solution feasible is that enclave has been widely existed in the mainstream system, such as ARM Trustzone, Intel SGX, RISC-V Keystone, etc.

With the help of secure enclave, it becomes possible to extend PRE to fit in more encryption schemes, including both symmetric and asymmetric encryption schemes. Note that symmetric encryption costs much less than public-key encryption, and thus the combination of symmetric encryption significantly improves the performance of the entire PRE scheme, which provides a wider range of potential applications.

### B. Contributions

In this work, we design a new PRE scheme based on secure enclave to satisfy the practical needs of PRE applications, and we name it as hPRESS. hPRESS achieves better performance and better features than existing cryptographic PRE schemes. To the best of our knowledge, hPRESS is the first unidirectional multi-hop PRE that supports constant decryption efficiency and constant ciphertext size without extensions. Our contributions are summarized as follows.

- First, we present hPRESS, a generic PRE construction scheme based on secure enclave. hPRESS is the first unidirectional multi-hop PRE with constant decryption efficiency and constant ciphertext size without extensions. It also supports ciphertext transformation between arbitrary encryption schemes. The security of hPRESS is guaranteed by both secure hardware and cryptography instead of pure cryptography.
- Second, we extend the original primitives of PRE to fit in our hPRESS design. We expand the support for various public-key and symmetric encryption into our new primitives, which provides new features and to satisfy the performance demand for PRE in practice, and widely expands the spectrum of PRE's application. Note that all the existing PRE schemes can only transform ciphertexts either between public-key encryptions or between symmetric key encryptions [3], while our new primitives break the gap. We also provide a detailed security analysis of our hPRESS with formal game-based security proof. We prove that hPRESS achieves chosen-ciphertext-attack security.
- Third, we implement an instance of hPRESS based on Intel SGX. We evaluate the prototype, with extensive experiments and the results demonstrate that our proposal achieves 10 times better performance than existing PRE schemes with improved scalability.

### C. Organization

The rest of this paper consists of the following sections. The relevant background of PRE and the definitions of PRE are presented in Section II, and the security models are also described. We describe our adversary model in Section III, and the design description of hPRESS and SGX-based instance are proposed in Section IV. The detailed security analysis of hPRESS is described in Section V. The implementation of hPRESS and related performance evaluation are provided in Section VI. Related works of our paper lie in Section VII. Finally, our conclusion is given in Section VIII.

## II. Primitives and Definitions

In this section, we present our new primitives and definitions of PRE. We also briefly review the security models of PRE, which will be used in the secondary proofs of hPRESS.

### A. New Definitions of PRE

We focus on the constructing a generic unidirectional, multi-hop proxy re-encryption. We extend the definition of PRE. Previous PRE definitions only support ciphertext transformations in the same encryption scheme, while our new definition extends to the symmetric encryption schemes, and supports transformations between symmetric and public-key encryption schemes, only if their message spaces are compatible.

A generic unidirectional multi-hop PRE scheme consists of the following five algorithms as shown in Fig. 1.

- $\text{KeyGen}(1^\lambda) \rightarrow (pk, sk)$ Given the security parameters $1^\lambda$, the key generation algorithm KeyGen outputs an encryption key $pk$ and decryption key $sk$. This algorithm is executed by both the delegator and delegatee, and they can choose different encryption schemes, e.g., one chooses

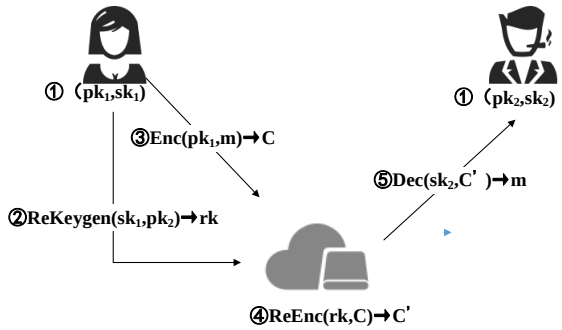



Fig. 1: The Process of a Proxy Re-Encryption Scheme.

a symmetric key encryption scheme and the other chooses a public-key encryption scheme. Note that for public key encryption, $pk$ and $sk$ are the corresponding public key and secret key, respectively, and $pk$ and $sk$ are both the symmetric key for symmetric key encryption. Note that when $pk$ in the symmetric encryption scheme is used for encryption, the actual symmetric key is adopted, whereas when $pk$ is published, the index of the actual symmetric key is published instead. In the following algorithms, we implicitly set $pk = sk$ for symmetric key encryption.

- $\text{REKEYGEN}(pk_1, sk_1, pk_2, sk_2) \rightarrow (rk_{1\rightarrow 2})$ Given key pairs of both the delegator and delegatee $(pk_1, sk_1, pk_2, sk_2)$, the re-encryption key generation algorithm REKEYGEN outputs a unidirectional re-encryption key $rk_{1\rightarrow 2}$. Note that $pk_1$ and $sk_2$ are not mandatory-required in some concrete schemes, we keep them here for generalization.
- $\text{ENC}(pk, m) \rightarrow \texttt{CT}$ Given an encryption key $pk$ generated from KEYGEN and a message $m$ from the message space $\mathcal{M}$, the encryption algorithm ENC outputs a ciphertext CT.
- $\text{REENC}(rk_{1\rightarrow 2}, \texttt{CT}_1) \rightarrow \texttt{CT}_2$ Given a re-encryption key $rk_{1\rightarrow 2}$ and a ciphertext $\texttt{CT}_1$, the re-encryption algorithm REENC outputs a ciphertext $\texttt{CT}_2$ or the error symbol $\perp$.
- $\text{DEC}(sk, \texttt{CT}) \rightarrow m$ Given a decryption key $sk$ and a ciphertext CT, the decryption algorithm DEC outputs a message $m$ or the error symbol $\perp$.

**Correctness** : We say that a generic unidirectional multi-hop PRE scheme (KEYGEN, REKEYGEN, ENC, REENC, DEC) is correct only if the following conditions are satisfied for $(pk_i, sk_i) \leftarrow \text{KEYGEN}(1^\lambda)$ and $m \in D$:

- $\text{DEC}(sk, \text{ENC}(pk, m)) = m$;
- $\text{DEC}(sk_i, \text{REENC}(rk_{(i-1)\rightarrow i}, \text{REENC}(rk_{(i-2)\rightarrow(i-1)}, \cdots (\text{REENC}(rk_{1\rightarrow 2}, \text{ENC}(pk_1, m)))))) = m$, where $rk_{j\rightarrow j'} = \text{REKEYGEN}(pk_j, sk_j, pk'_j, sk'_j)$, $j, j' \in \{1, 2, \cdots, i\}$.

### B. Security Model

Following the above security model for PRE, we define the security model of hPRESS through a game interacted between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

- **Phase 1:** $\mathcal{A}$ is allowed to adaptively query the five oracles to $\mathcal{C}$.
  - **Key generation oracle** $\mathcal{O}_{pk}$: Given a name of encryption scheme from $\mathcal{A}$, $\mathcal{C}$ firstly runs the corresponding key generation algorithm to get $(pk, sk)$, and then returns $pk$ to $\mathcal{A}$. If the scheme is a symmetric key encryption, $\mathcal{A}$ is given the index of $sk$ instead of $pk$.
  - **Key corruption oracle** $\mathcal{O}_{sk}$: Given $pk$ generated from $\mathcal{O}_{pk}$ from $\mathcal{A}$, $\mathcal{C}$ responds with the corresponding $sk$.
  - **Re-encryption key generation oracle** $\mathcal{O}_{rk}$: Given $(pk, pk')$ from $\mathcal{A}$, $\mathcal{C}$ returns the corresponding re-encryption key $rk_{pk\rightarrow pk'} = \text{REKEYGEN}(pk, sk, pk', sk')$, where $sk$ and $sk'$ are the decryption keys respectively corresponding with $pk$ and $pk'$, and $pk$ is generated from $\mathcal{O}_{pk}$.
  - **Re-encryption oracle** $\mathcal{O}_{re}$: Given $(pk, pk', \texttt{CT})$, $\mathcal{C}$ returns the corresponding transformed ciphertext $C' = \text{REENC}(\text{REKEYGEN}(pk, sk, pk', sk'), \texttt{CT})$, where $\text{DEC}(sk, \texttt{CT}) = \text{DEC}(sk', \texttt{CT}')$, and $sk$ and $sk'$ are the decryption keys respectively corresponding with $pk$ and $pk'$, and $pk$ is generated from $\mathcal{O}_{pk}$.
  - **Decryption oracle** $\mathcal{O}_{de}$: Given $(pk, \texttt{CT})$ from $\mathcal{A}$, $\mathcal{C}$ returns the corresponding message $\text{DEC}(sk, \texttt{CT})$, where $(pk, sk)$ are generated from $\mathcal{O}_{pk}$.
- **Challenge Phase:** Given two messages $m_0^*, m_1^*$ of equal length and a series of encryption keys $pk_1^*, pk_2^*, \cdots, pk_i^*$ generated from $\mathcal{O}_{pk}$ from $\mathcal{A}$, $\mathcal{C}$ encrypts $m_b^*$ as the challenge ciphertext $\texttt{CT}^*$ to $\mathcal{A}$, where $b$ is chosen randomly from $\{0, 1\}$.
  $\texttt{CT}^* = \text{REENC}(rk^*_{(i-1)\rightarrow i}, \text{REENC}(rk^*_{(i-2)\rightarrow(i-1)}, \cdots (\text{REENC}(rk^*_{1\rightarrow 2}, \text{ENC}(pk_1^*, m_b^*)))))$.
  Note that $pk_i^*$ cannot be corrupted. We say a key pair $(pk, sk)$ is corrupted when it is directly corrupted by $\mathcal{O}_{sk}$, or when it has been queried to $\mathcal{O}_{rk}$ with another corrupted key $pk'$, which we call an indirect corruption.
- **Phase 2:** Phase 2 is almost the same as Phase 1, except that the following restricts should be added to the original oracles.
  - $\mathcal{O}_{rk}$: $pk$ cannot be a derivative key of $pk_i^*$, if $pk'$ is a corrupted key. We say $pk'$ is a derivative key of $pk^*$, if $(pk, pk')$ is queried to $\mathcal{O}_{rk}$, or if $pk'$ is a derivative key of $pk''$ and $pk''$ is a derivative key of $pk^*$.
  - $\mathcal{O}_{re}$: If $\mathcal{A}$ inputs $(pk, pk', \texttt{CT})$ to $\mathcal{O}_{re}$, where $(pk, \texttt{CT})$ is a derivative of $(pk_i^*, \texttt{CT}^*)$, and $pk'$ is a corrupted key, $\mathcal{C}$ returns $\perp$ instead. We say $(pk', \texttt{CT}')$ is a ciphertext derivative of $(pk^*, \texttt{CT}^*)$ when $\texttt{CT}'$ is the output of $\mathcal{O}_{re}(pk^*, pk', \texttt{CT}^*)$, or $\text{REENC}(rk_{pk^*\rightarrow pk'}, \texttt{CT}^*)$, where $rk_{pk^*\rightarrow pk'}$ is generated by $\mathcal{O}_{rk}$, or if $(pk', \texttt{CT}')$ is a ciphertext derivative of $(pk'', \texttt{CT}'')$ and $(pk'', \texttt{CT}'')$ is a ciphertext derivative of $(pk^*, \texttt{CT}^*)$.
  - $\mathcal{O}_{dec}$: If $\mathcal{A}$ inputs $(pk, \texttt{CT})$ to $\mathcal{O}_{dec}$, where $(pk, \texttt{CT})$ is a ciphertext derivative of $(pk_i^*, \texttt{CT}^*)$, $\mathcal{C}$ returns $\perp$ instead.
- **Guess Phase:** Finally, $\mathcal{A}$ outputs the guess result $b' \in \{0, 1\}$ and we say $\mathcal{A}$ wins the game if $b = b'$.
  We refer to this kind of $\mathcal{A}$ as an IND-CCA adversary to unidirectional multi-hop PRE. We define the adversary $\mathcal{A}$'s advantage in attacking the scheme $\varepsilon$ as a function of the security parameter $k$, which is the input given to the challenger $\mathcal{C}$.

$Adv_{\varepsilon,\mathcal{A}}(k) = |Pr[b = b'] - 1/2|$

The probability is over the random bits used by both $\mathcal{A}$ and $\mathcal{C}$.

Based on the above game, we can define chosen ciphertext security for a PRE scheme. As usual, we say a function $g : \mathcal{R} \rightarrow \mathcal{R}$ is negligible if for any $d > 0$ we have $|g(k)| < 1/k^d$ for a sufficiently large $k$.

*Definition 1:* We say that a PRE system $\varepsilon$ is secure against an adaptive chosen ciphertext attack if for any polynomial-time IND-CCA adversary $\mathcal{A}$, the function $Adv_{\varepsilon,\mathcal{A}}(k)$ is negligible. In other words, we say $\varepsilon$ is IND-CCA secure.

## III. Adversary Model

We consider a powerful adversary who might control any software stacks outside the secure enclave, for example, hypervisor and OS if Intel SGX enclaves are adopted. The adversary is also able to control the network packets, such as copying, replacing or modifying the original packets. In other words, the adversary who is in possession of the hardware can monitor and tamper with all the input to the secure enclaves and the corresponding outputs. The encryption and signature schemes adopted in this paper are required to be CCA-secure to prevent the output ciphertexts and signatures from leaking information. However, we assume that the adversary is unable to physically open or manipulate the secure enclave packages where some sensitive messages reside in. It is already revealed that some secure enclave techniques suffer from side-channel or fault injection attacks [4], [5], which are outside the scope of this paper.

## IV. Design

Our new PRE definition covers almost all possible PRE situations as it combines most demanded features of a practical PRE scheme. However, it is hard to construct a specific PRE scheme fitting in the definition with pure cryptography. A better solution is to adopt secure hardware enclave technique such as Intel SGX to simplify the design. In this section, we propose a generic construction for unidirectional multi-hop PRE scheme by using secure enclave to show that the definition is feasible, and we also construct an instance of `hPRESS` using SGX.

### A. Overview

To design a unidirectional multi-hop PRE, forklore proposals aim to divide the original key into two pieces or more. For example, the delegator divides the decryption key $sk_1$ into two subkeys, $sk_{1,1}$, $sk_{1,2}$, and then encrypts the message $m$ with $sk_{1,2}$, which outputs a ciphertext $C_{1,2}$. $C_{1,2}$ and $sk_{1,1}$ are delivered to the proxy. The proxy re-encrypts $C_{1,2}$ with $sk_{1,1}$ and the encryption key of the delegatee $pk_2$, then outputs $C'$ and $C''$ respectively. The proxy sends $C_{1,2}$, $C'$ and $C''$ to the delegatee. The delegatee decrypts the three ciphertexts and obtains the original message $m$. It is obvious that in this kind of proposal, the proxy is required to execute more modular computations, and the delegatee is required to decrypt three times, which results in a linear growth of the ciphertext extensions.

To address this problem, we reduce the time of both re-encryption and decryption of PRE to only once. It seems impossible at first sight, because the entire $m$ inevitably appears in the intermediate calculations, which is unacceptable. However, with the help of secure hardware enclave, the above idea becomes practical. We utilize the secure execution environment supported by hardware security to keep the message secret. For example, one way is to decrypt with $sk_1$ and then encrypt with $pk_2$ in the trusted execution environment.

In this case, the security guarantee of the secret messages is handled by the combination of secure hardware enclave and secure cryptographic schemes, not pure cryptography. Secure enclave can efficiently defend against active fault attacks and passive side-channel attacks, such as power and electromagnetic attacks except for cache attacks. We utilize this feature to protect our scheme from these side-channel attacks and fault attacks. Although existing enclave techniques such as SGX still suffer from micro-architectural attacks, our scheme is scalable for future enclave techniques. Once hardware companies present a new secure enclave technique, it is easy to adopt it in our design to prevent these attacks.

### B. Architecture

To show our idea is practical, we give a detailed description of `hPRESS` architecture. In our design, the proxy holds a server, and each delegator or delegatee holds a client. These entities cooperate to accomplish the hardware-based PRE. The proxy server has a PRE enclave and two tables to store the keys. PRE Enclave aims to provide a trusted execution environment for decryption. The two tables, PK Table and SK Table, are used to store the delegator and delegatee's encryption keys and decryption keys respectively. Note that SK Table need to be sealed in ciphertext, while PK Table can be published in plaintext. The client program of the delegator and delegatee is simpler. It accomplishes the authentication between the proxy and the delegator, and on-duty of delivering the decryption key of the delegator to the proxy secretly.

`hPRESS` works as follows. The proxy server and the delegator client programs generate verification / signing key pairs for a cryptographic signature scheme respectively. The verification keys are published, and the signing keys are sealed with the sealing keys and kept locally in non-volatile storage. After the delegator initializes the client program, the client enclave starts to attest and establish a secure channel with the PRE Enclave run by the proxy. The client enclave receives the attestation feedback from the proxy, which indicates the remote attestation is finished and the secure channel is established with a key pair generated inside the enclave. The delegator sends the decryption key to the proxy over this channel. PRE Enclave receives the key, seals and stores it in the decryption key table. Note that the decryption key here is the private key in public-key cryptography and the secret key in symmetric cryptography.

The data producer provides the data from the delegator to the delegatee, which is originally encrypted with the encryption key of the delegator, and the encrypted data is sent to the proxy. When the delegator queries to re-encrypt the data

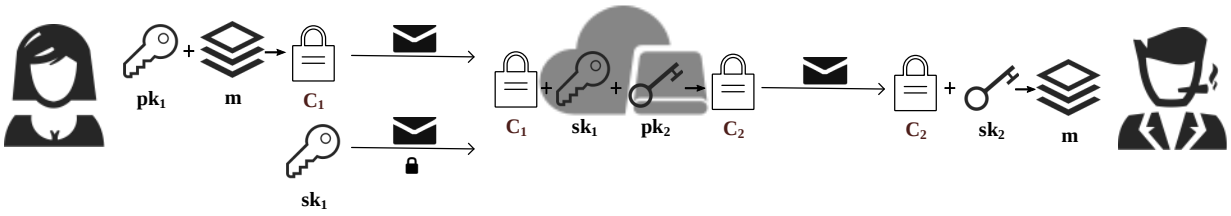



Fig. 2: Overview of hPRESS.

to the delegatee, the client program of the delegator signs the metadata and the labels of the target delegatees using the signing key generated in the setup phase. The metadata includes the necessary information for data re-encryption, such as file path, offset and so on. Then the delegator sends the message and the signature together to the proxy as a "re-encryption key". When the proxy receives the message from the delegator, it first verifies the attached signature to prevent a forgery attack. If verification succeeds, PRE Enclave reads the decryption key of the delegator from the SK Table, decrypts, and get the plaintext data. Then it reads the encryption key of the delegatee from the PK Table, encrypts, and gets the re-encrypted ciphertext. Finally, the proxy signs the result, and sends the new ciphertext to the delegatee. Figure 2 shows an basic re-encryption process in our scheme.

### C. Intel SGX

As secure enclave techniques significantly differ from one another, it is hard to present a generic model which is applicable to all scenarios. Thus we construct hPRESS with one of the existing secure enclave techniques as an instance. Note that in our hPRESS design, we need the secure enclave to support the feature of attestation, and thus we adopt Intel SGX to construct the hPRESS instance. The implementation of this SGX-based instance refers to the Iron of Fisch et al. [6], which tried to use Intel SGX to propose an advanced Functional Encryption (FE) scheme.

We first briefly introduce the SGX features relevant to our design. The two most important features of Intel SGX are isolation / sealing and attestation, and we also introduce the side channel vulnerability of SGX as well as the hardware model.

*1) Isolation / Sealing:* Isolation / Sealing guarantees the security and integrity of a single enclave. There is a hardware-guarded memory area of SGX called EPC (Enclave Page Cache). At the beginning of the initialization of an enclave program, the code and data are copied from external memory to EPC pages. Then the initialization process hashes the context on each page. The hash output is called MRENCLAVE, which is stored in EPC. The entry of this enclave is not permitted until the initialization process finishes, in which way, the enclave identity is built. SGX hardware ensures that only the pages associated with the same identity can access each other.

Before the program finishes running inside this enclave, the code and data are protected from the external processes. When the enclave stops, it seals its data with a Seal Key. The Seal Key is retrieved from the Root Seal Key using the EGETKEY instruction, and the Root Seal Key is embedded in every SGX-supported processor during manufacturing. The enclave encrypts and authenticates its data using this Seal Key, and passes the encrypted data to the external untrusted environment. When this enclave is restarted, it can still recover the data stored in the untrusted memory.

*2) Attestation:* Attestation guarantees security and integrity when two SGX processes communicate with each other. SGX attestation consists of two forms: Local Attestation and Remote Attestation. Local attestation is the attestation process between two enclaves on the same platform. Two enclaves on the same platform generate a shared authentication key called the Report Key using the EGETKEY instruction. One of them calls the EREPORT instruction to hash its MRENCLAVE and metadata with the Report Key. Then it sends the report to the delegatee enclave. If the verification of the delegatee is successful, it proves that this enclave is expected to run on the same platform. Remote attestation [7] is the attestation process between an enclave and a third-party on different platforms. To implement remote attestation, the enclave has to locally attest to a special architectural enclave called Quoting Enclave (QE). After QE receives the report from the local enclave, it verifies and transforms the report into a quote by signing it with another key, which is called the Provisioning Key. The Provision Key is generated by an anonymous group signature scheme called Intel Enhanced Privacy ID (EPID). To verify the signature in a quote, the third-party has to contact the Intel Attestation Server.

*3) Side Channel Vulnerability:* One well-known issue of Intel SGX is that it is subject to side channel attacks [7]. Even if SGX enclaves are isolated, the malicious operating system still takes control of the system calls and the memory management, from which the adversary can easily obtain the control flow information of the program running in the enclave [5]. Side channel attacks remain a threat to SGX although researchers have present several solutions to make SGX resistant to side channel attacks, such as involving ORAM [8] or Intel TSX [9]. Moreover, it has been proven that the micro-architectural L1TF attacks can also be applied to SGX [4], which become another affective threat to Intel SGX.

In this work, we try to handle these side channel attacks. To deal with memory-based attacks, we adopt side-channel-resistant encryption schemes. In [5], the authors provided a list of encryption schemes resistant to control flow leakages in the OpenSSL and Libgcrypto Libraries. We adopt some of

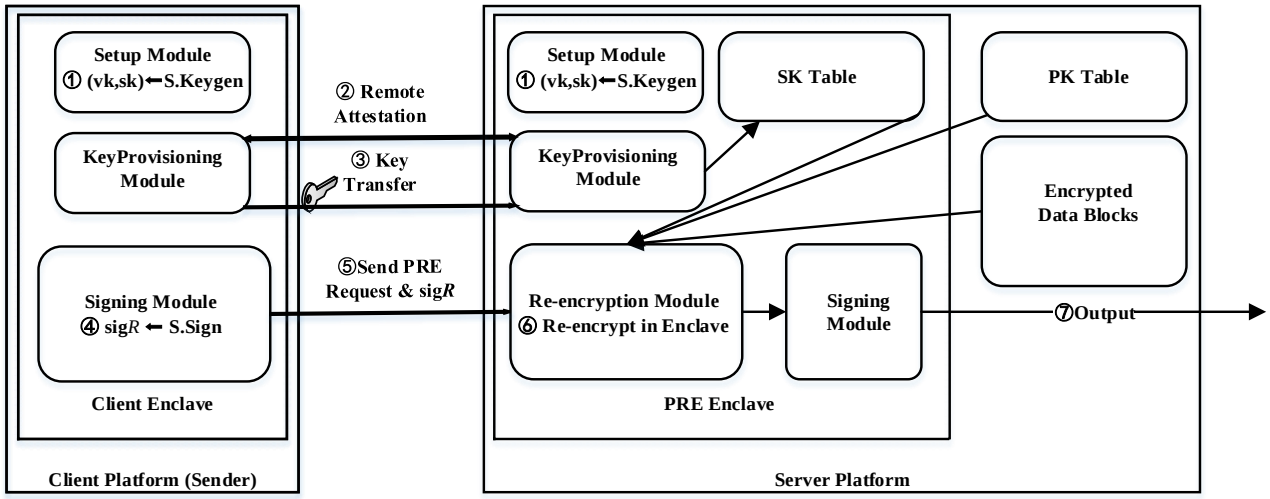



Fig. 3: Architecture of hPRESS.

these schemes to construct our design without breaking their side-channel resistance. As for L1TF attacks, they are micro-architectural vulnerabilities which can only be addressed by microcode patches. A possible solution is to verify the microcode version during the re-encryption.

*4) Hardware Primitives:* To formally describe our construction using Intel SGX, a theoretical hardware model is required. Following the system model of Iron in [6], we define a secure hardware model HW for probabilistic-polynomial-time programs Q consisting of the following interface: HW.Setup, HW.Load, HW.Run, HW.Quote, and HW.QuoteVerify. HW maintains two variables, $\mathtt{HW.sk}_{\mathtt{quote}}$ and $\mathtt{HW.sk}_{\mathtt{report}}$, and a table T consisting of enclave states indexed by enclave handles. $\mathtt{HW.sk}_{\mathtt{quote}}$ and $\mathtt{HW.sk}_{\mathtt{report}}$ are used for storing signing keys for reporting and quoting and T is used to manage the internal states of loaded enclave programs.

- HW.Setup takes as input a security parameter $\lambda$. Then it generates the secret keys $\mathtt{sk}_{\mathtt{quote}}$, $\mathtt{sk}_{\mathtt{report}}$, and stores them in $\mathtt{HW.sk}_{\mathtt{quote}}$, $\mathtt{HW.sk}_{\mathtt{report}}$ respectively. Finally, it generates and outputs public parameters $params$.
- HW.Load loads a program into an enclave. HW.Load takes as input a program Q and global parameters params. It creates an enclave first, loads Q, and generates a handler $hdl$ which is used to identify the enclave. It also initializes the entry $\mathtt{T}[\mathtt{hdl}] = \emptyset$.
- HW.Run runs an enclave program. It takes in a handle hdl corresponding to an enclave running program Q and an input in. It runs Q at state $\mathtt{T}[\mathtt{hdl}]$ with input in and records the output out. It updates $\mathtt{T}[\mathtt{hdl}]$ and outputs out.
- HW.Quote executes an enclave program, and also generates an attestation of a program's output out that can be publicly verified by a remote party. It takes as input a handle hdl corresponding to an enclave running a program Q, and an input in for Q. This algorithm has restricted access to the key $\mathtt{sk}_{\mathtt{quote}}$ which is used to sign messages. The algorithm executes and outputs the tuple $\mathtt{quote} = (\mathtt{mdhdl}, \mathtt{tagQ}, \mathtt{in}, \mathtt{out}, \sigma)$, where mdhdl is the metadata associated with the enclave, tagQ is a program tag for Q and $\sigma$ is a signature on $(\mathtt{mdhdl}, \mathtt{tagQ}, \mathtt{in}, \mathtt{out})$.
- HW.QuoteVerify is the quote verification algorithm. It takes params and $\mathtt{quote} = (\mathtt{mdhdl}, \mathtt{tagQ}, \mathtt{in}, \mathtt{out}, \sigma)$ as input. It outputs 1 if the signature verification of $\sigma$ succeeds and 0 otherwise.

### D. Instance Constructions

To formally analyze our scheme, we present a theoretical description of the SGX-based hPRESS instance with the help of the above hardware primitives.

In this hPRESS instance, the delegator D, and the proxy P both get access to the instance of HW, while the delegatee D′ is not required to support the enclave technique. (Here refers to SGX.) E denotes the CCA-secure encryption schemes that the delegator and delegatee adopt. S denotes a secure signature scheme. We present the cryptographic description of this hPRESS instance, and explain the following primitives in hPRESS: KeyGen, ReKeyGen, Encrypt, Reencrypt and Decrypt. We prove that hPRESS fits well in the PRE primitives, and satisfies the PRE correctnesses.

*1)* KeyGen*:* D and D′ first run the KeyGen algorithm of one or two encryption schemes to generate their encryption/decryption key pairs respectively $(pk_1, sk_1)$ and $(pk_2, sk_2)$. Then D and P run $\mathtt{HW.Setup}(1^\lambda)$ for their HW instances respectively, and record the corresponding output parameters params.

Both D and P run their key generation algorithms (Algorithm 1 and 2):

$\mathtt{hdl}_{\mathtt{D}} \leftarrow \mathtt{HW.load}(\mathtt{params}, \mathtt{Q}_{\mathtt{D.KeyGen}})$
and $\mathtt{hdl}_{\mathtt{P}} \leftarrow \mathtt{HW.load}(\mathtt{params}, \mathtt{Q}_{\mathtt{P.KeyGen}})$.

We define the KenGen algorithms of P and D ($\mathtt{Q}_{\mathtt{P.KeyGen}}$ and $\mathtt{Q}_{\mathtt{D.KeyGen}}$) as follows. Note that the measurement of P hs been hardcoded in the static data of D.

**Algorithm 1** Framework of $\mathtt{Q}_{\mathtt{D.KeyGen}}$.

**Input:** $(\text{“}\mathtt{init}\text{”}, 1^\lambda)$:
1: Run $(vk_D, sk_D) \leftarrow \mathtt{S.KeyGen}(1^\lambda)$.
2: Update $sk_D$, and **return** $vk_D$.
**Input:** $(\text{“}\mathtt{provision}\text{”}, \mathtt{quote_P}, pk_{hw})$:
3: Parse $\mathtt{quote_P} = (\mathtt{md_{hdl}}, \mathtt{tag_Q}, \mathtt{in}, \mathtt{out}, \sigma)$;
4: Check if $\mathtt{tag_Q} = \mathtt{tag_P}$. If not, **return** $\perp$.
5: Run $b \leftarrow \mathtt{HW.QuoteVerify}(\mathtt{params}, \mathtt{quote_P})$.
6: If $b = 0$, **return** $\perp$.
7: Get the delegator's decryption key $sk_1$.
8: Compute $ct_{sk} = \mathtt{E.Enc}(pk_{hw}, sk_1)$;
9: and $\sigma_{sk} = \mathtt{S.Sign}(sk_D, ct_{sk})$.
10: **return** $ct_{sk}$, $\sigma_{sk}$.

**Algorithm 2** Framework of $\mathtt{Q}_{\mathtt{P.KeyGen}}$.

**Input:** $(\text{“}\mathtt{init}\text{”}, 1^\lambda)$:
1: Run $(vk_P, sk_P) \leftarrow \mathtt{S.KeyGen}(1^\lambda)$.
2: Update $sk_P$, and **return** $vk_P$.
**Input:** $(\text{“}\mathtt{attestation}\text{”}, \mathtt{quote_D}, 1^\lambda)$:
3: Parse $\mathtt{quote_D} = (\mathtt{md})_{\mathtt{hdl}}, \mathtt{tagQ}, \mathtt{in}, \mathtt{out}, \sigma)$.
4: Check that $\mathtt{tag_Q} = \mathtt{tag_D}$.
5: If not, **return** $\perp$.
6: Run $b \leftarrow \mathtt{HW.QuoteVerify}(\mathtt{params}, \mathtt{quote_D})$.
7: If $b = 0$, **return** $\perp$.
8: Run $(pk_{hw}, sk_{hw}) \leftarrow \mathtt{PKC.KeyGen}(1^\lambda)$.
9: Update $sk_{hw}$, and **return** $pk_{hw}$.
**Input:** $(\text{“}\mathtt{provision}\text{”}, \mathtt{ct_{sk}}, \sigma_{sk})$:
10: Verify the signature, $b \leftarrow \mathtt{S.Verify}(vk_D, \sigma_{sk}, ct_{sk})$.
11: If $b = 0$, **return** $\perp$.
12: Run $m \leftarrow \mathtt{E.Dec}(sk_{hw}, ct_{sk})$.
13: Parse $m = (sk_1)$.
14: Add $sk_1$ to $\mathtt{state}$.

Generate keys as well as the quote of the delegator enclave used in key provisioning.

$vk_P \leftarrow \mathtt{HW.R}(\mathtt{hdl_P}, (\text{“}\mathtt{init}\text{”}, 1^\lambda))$,

$(vk_D, \mathtt{quote_D}) \leftarrow \mathtt{HW.R\&Q}(\mathtt{hdl_D}, (\text{“}\mathtt{init}\text{”}, 1^\lambda))$.

After the proxy receive the quoted message from the delegator, run

$(\mathtt{quote_P}) \leftarrow \mathtt{HW.R\&Q}(\mathtt{hdl_P}, (\text{“}\mathtt{attestation}\text{”}, \mathtt{quote_D}, 1^\lambda))$.

The delegator then runs

$(ct_{sk}, \sigma_{sk}) \leftarrow \mathtt{HW.R}(\mathtt{hdl_D}, (\text{“}\mathtt{provision}\text{”}, \mathtt{quote_P}, pk_{hw}))$.

After receiving the feedback, the proxy runs

$\mathtt{HW.R}(\mathtt{hdl_P}, (\text{“}\mathtt{provision}\text{”}, ct_{sk}, \sigma_{sk}))$.

*2)* ENC*:* The data producer or the delegator encrypts a secret message $m$ with $pk_1$, and sends the ciphertext to the proxy.

$ct_1 \leftarrow \mathtt{Enc}(pk_1, m)$.

*3)* REKEYGEN*:* D runs the signing algorithm in the enclave (Algorithm 3):

$(\{md_m, rid\}, \sigma_{req}) \leftarrow \mathtt{HW.R}(\mathtt{hdl_D}, (\text{“}\mathtt{sign}\text{”}, md_m, rid))$.

$(\{md_m, rid\}, \sigma_{req})$ is generated as “re-encryption key” of hPRESS.

*4)* REENC*:* The re-encryption program of P is defined as follows.

The proxy runs Algorithm 4:

**Algorithm 3** Framework of $\mathtt{Q}_{\mathtt{D.Sign}}$.

**Input:** $(\text{“}\mathtt{sign}\text{”}, md_m, rid)$:
1: Compute $\sigma_{req} \leftarrow \mathtt{S.Sign}(sk_D, \{md_m, rid\})$.
2: **return** $\{md_m, rid\}, \sigma_{req}$.

$(ct_2, \sigma_{re}) \leftarrow \mathtt{HW.R}(\mathtt{hdl_P}, (\text{“}\mathtt{reencrypt}\text{”}, \{md_m, rid\}, \sigma_{req}))$.

The output of REENCRYPT is sent to the delegatee.

**Algorithm 4** Framework of $\mathtt{Q}_{\mathtt{P.Reencrypt}}$.

**Input:** $(\text{“}\mathtt{reencrypt}\text{”}, ct_{req}, \sigma_{req})$:
1: Verify the signature, $b \leftarrow \mathtt{S.Verify}(vk_D, \sigma_{req}, \{md_m, rid\})$.
2: If $b = 0$, **return** $\perp$.
3: Parse $md_m$ and $rid$ in $m_{req}$.
4: Read $sk_1$ in $\mathtt{state}$.
5: Read $ct_1$ and $pk_2$.
6: Run $m \leftarrow \mathtt{E.Dec}(sk_1, ct_1)$ to calculate the plaintext data block $m$.
7: Compute $ct_2 \leftarrow \mathtt{E.Enc}(pk_2, m)$,
8: and $\sigma_{re} \leftarrow \mathtt{S.Sign}(sk_P, ct_2)$.
9: **return** $ct_2$, $\sigma_{re}$.

*5)* DEC*:* The delegatee receives the re-encrypted message $ct_2$ and the corresponding signature $\sigma_{re}$, and then verify $\sigma_{re}$.

$b \leftarrow \mathtt{S.Verify}(vk_P, \sigma_{re}, ct_2)$.

If $b = 0$, the decryption algorithm fails, else the delegatee decrypts the re-encrypted message to get the secret message $m$.

$m \leftarrow \mathtt{E.Dec}(sk_2, ct_2)$

## V. SECURITY ANALYSIS

In this section, we combine the primitives with the definition to complete our security analysis. First we prove that our scheme satisfies the correctnesses mentioned in Section II, and with the help of sequence games, we show how to prove hPRESS is CCA-secure. Note that the security analysis of symmetric encryption is similar to that of public-key encryption, and thus we omit the security analysis in the symmetric encryption case. Note that our security analysis is a generic one, although we take SGX as an instance here for simplicity. Similar analysis procedures can be applied to hPRESS based on other secure enclave techniques as well.

### A. Correctnesses Proof

The correctness of our new PRE definition is discussed in Section II:

$$\text{DEC}(sk, \text{ENC}(pk, m)) = m \tag{1}$$

$$\text{DEC}(sk_i, \text{REENC}(rk_{(i-1)\to i}, \text{REENC}(rk_{(i-2)\to(i-1)}, \cdots$$

$$\cdots(\text{REENC}(rk_{1\to 2}, \text{ENC}(pk_1, m)))))) = m \tag{2}$$

where $rk_{j\to j'} = \text{REKEYGEN}(pk_j, sk_j, pk'_j, sk'_j)$

It is obvious that the requirement 1 is obtained directly from the correctness of the underlying encryption scheme. As for the requirement 2, we have introduced that in our proposal the re-encryption is a decryption and encryption in

the trusted execution environment supported by SGX, which means that $\text{REENC}(rk_{i\to j}, \text{ENC}(pk_i, m)) = \text{ENC}(pk_j, m)$. Hence the equation of the requirement 2 can be simplified to $\text{DEC}(sk_i, \text{ENC}(pk_i, m)) = m$, which is obviously correct under the requirement 1.

### B. Security Proofs

*Theorem 1:* If the underlying encryption scheme is a CCA-secure encryption scheme, and HW is a secure hardware system, our proposal is CCA secure.

*Proof:* Our goal is to prove that if there is an adversary $\mathcal{A}$ that can successfully break CCA security on our proposal, we can build an algorithm breaking any CCA-secure encryption scheme. The new game sequence of $G_0$ and $G_1$ is described as follows. $G_0$ is the main game modified from the security model of our new PRE definition. $G_1$ is the security game of a CCA-secure encryption scheme contain oracles of key generation, encryption and decryption.

- **Phase 1:** $\mathcal{A}$ is allowed to adaptively query the PRE oracles – $\mathcal{O}_{pk}$, $\mathcal{O}_{sk}$, $\mathcal{O}_{rk}$, $\mathcal{O}_{re}$ and $\mathcal{O}_{de}$. However, these oracles need to be modified in some degree.
  - **Uncorrupted key generation oracle** $\mathcal{O}_{pk}$: Given a name of encryption scheme from $\mathcal{A}$, $\mathcal{C}$ chooses a $\theta \in \{0,1\}$ with $\Pr[\theta = 1] = \delta$. If $\theta = 1$, $\mathcal{C}$ normally obtains a new key pair $(pk, sk)$, and $\mathcal{A}$ is given $pk$. If $\theta = 0$, $\mathcal{C}$ invokes $\mathcal{O}_{pk}$ in $G_1$, gets a public key $pk$ without knowing the pairing private key, and returns $pk$ to $\mathcal{A}$. If the scheme is a symmetric key encryption, $\mathcal{A}$ is given the index of $sk$ instead of $pk$.
  - **Key corruption oracle** $\mathcal{O}_{sk}$: Given $pk$ from $\mathcal{A}$ which is generated from $\mathcal{O}_{pk}$, $\mathcal{C}$ returns the corresponding $sk$ if $\theta_{pk} = 1$. Otherwise if $\theta_{pk} = 0$, $G_0$ aborts because $\mathcal{C}$ does not know $sk$ either, and returns $\perp$ to $\mathcal{A}$.
  - **Re-encryption key generation oracle** $\mathcal{O}_{rk}$: Given $(pk, pk')$ from $\mathcal{A}$, $\mathcal{C}$ returns the corresponding re-encryption key $rk_{pk\to pk'} = \text{REKEYGEN}(pk, sk, pk', sk')$, where $sk$ and $sk'$ are the decryption keys respectively corresponding with $pk$ and $pk'$, and $pk$ is generated from $\mathcal{O}_{pk}$. Assume that HW is secure, then the re-encryption key is actually a signature, which does not include any information about the message and the secret keys (in Section IV), so we replace it with a random oracle that output random message as long as a legal signature, which has no business to the secrets.
  - **Re-encryption oracle** $\mathcal{O}_{re}$: $\mathcal{A}$ inputs $(pk, pk', \texttt{CT})$, and $pk$ and $pk'$ are both generated by KEYGEN. $\mathcal{C}$ returns the re-encrypted ciphertext $\texttt{CT}' = \text{REENC}(\text{REKEYGEN}(sk, pk'), \texttt{CT})$. If $(pk, \texttt{CT})$ does not match, $\mathcal{C}$ returns $\perp$ instead. Note that if $\theta_{pk} = 1$, the output $\texttt{CT}'$ is computed locally in the secure hardware model. $\mathcal{C}$ decrypts CT, and then encrypts the plaintext $m$ with the underlying encryption CCA-secure scheme to get $\texttt{CT}'$. Otherwise if $\theta_{pk} = 0$, $\mathcal{C}$ first invokes the $\mathcal{O}_{dec}$ in $G_1$, and encrypts the result using $pk'$.
  - **Decryption oracle** $\mathcal{O}_{de}$: Given $(pk, C)$ from $\mathcal{A}$, where $pk$ is generated by by KEYGEN, $\mathcal{C}$ returns the plaintext $\text{DEC}(sk, \texttt{CT})$. If $(pk, \texttt{CT})$ does not match, $\mathcal{C}$ returns $\perp$ instead. Note that if $\theta_{pk} = 1$, $\mathcal{C}$ decrypts CT with the corresponding decryption scheme. Otherwise if $\theta_{pk} = 0$, $\mathcal{C}$ invokes the $\mathcal{O}_{dec}$ in $G_1$, and returns the result to $\mathcal{A}$.
- **Challenge Phase:** $\mathcal{A}$ sends two messages $m_0^*, m_1^*$ of equal length and a public key $pk^*$ to $\mathcal{C}$. The constraints about input derivatives are still in effect. The difference is that, if $\theta_{pk^*} = 1$, $G_0$ aborts because $pk^*$ cannot help $\mathcal{C}$ to win $G_1$, the game sequence becomes meaningless. Otherwise if $\theta_{pk^*} = 0$, $\mathcal{C}$ forwards $m_0^*, m_1^*$ and $pk^*$ to the challenge oracle of $G_1$. The challenger of $G_1$ encrypts $m_b^*$ as the challenge ciphertext $\texttt{CT}^*$, where $b$ is chosen randomly from $\{0,1\}$. $\mathcal{C}$ receives the output $\texttt{CT}^*$ and then forwards it back to $\mathcal{A}$.
- **Phase 2:** Phase 2 is almost the same as Phase 1, however, the changes mentioned in the security model still should be added to these five oracles.
- **Guess Phase:** Finally, $\mathcal{A}$ outputs the guess result $b' \in \{0,1\}$ and we say $\mathcal{A}$ wins the game if $b = b'$. We refer to this kind of $\mathcal{A}$ as an IND-CCA adversary to our generic PRE proposal. We define the adversary $\mathcal{A}$'s advantage in attacking our scheme as a function of the security parameter $k$, which is the input given to the challenger $\mathcal{C}$.
  $Adv_{\mathcal{A}}(k) = |Pr[b = b'] - 1/2|$
  The probability is over the random bits used by both $\mathcal{A}$ and $\mathcal{C}$.

The trick is that, after $\mathcal{C}$ receives the guess result from $\mathcal{A}$, $\mathcal{C}$ forwards it to the challenger of $G_1$. If $\mathcal{A}$ has an non-negligible advantage $\epsilon$ to win $G_0$, then $\mathcal{C}$ has an non-negligible advantage to win $G_1$. The proof of claim is concise. Suppose that $\mathcal{A}$ makes a total of $q_{sk}$ key corruption oracle $\mathcal{O}_{sk}$ queries. Then the probability that $G_0$ does not abort in Phases 1 or 2 is $\delta^{q_{sk}}$. The probability that $G_0$ does not abort in the challenge phase is $(1-\delta)$. Therefore, the probability that $G_0$ does not abort during the simulation is $\delta^{q_{sk}}(1-\delta)$. This value is maximized at $\delta_{opt} = 1 - 1/(q_{sk}+1)$. Using $\delta_{opt}$, the probability that $G_0$ does not abort is at least $1/e(q_{sk}+1)$. This shows that $\mathcal{C}$ has an advantage of $1/e(q_{sk}+1)$ in $G_1$, which is non-negligible. The result means that $\mathcal{C}$ breaks the CCA security model. It is obviously impossible. Hence we can conclude that the assumption $\mathcal{A}$ can break the security of hPRESS is wrong.

Thus we complete the proof. ■

## VI. IMPLEMENTATION AND EVALUATION

### A. Implementation

We implement a prototype of hPRESS based on Intel SGX, which is developed using C and C++. We adopt the Intel (R) SGX SDK 2.1.102.43402 for Linus OS[1]. Both of the delegator and the proxy need to support SGXSSL, which is an extension of the OpenSSL Library in the SGX environment. In this paper, we use OpenSSL 1.1.0g.

[1]In Linux, all the enclaves are built as shared objects (.*so*). Trusted function calls named ECALLs are always executed in the enclaves. ECALLs are called from the untrusted applications outside the enclaves. Untrusted function calls named OCALLs are defined by the application. It calls from within a trusted enclave to the untrusted area.

*1) Proxy Server:* The proxy loads the enclave program `PREEnclave.so`. Once the PRE enclave is initialized, it runs the ECALL function `signkey_generate()` to generate an ECDSA key pair, which is used to sign the re-encryption results. It keeps the signing key secretly, and at the same time publishes the corresponding verification key. Then the proxy initializes the SK Table in the trusted memory region and the PK Table in the untrusted memory region. In another case, when the proxy restarts the enclave, it loads the key pairs and the SK Table which have already been generated and stored in the encrypted file, instead of a redundant initialization step. Next, the proxy listens to the network connection from the delegator on a remote platform. When the proxy gets a connection from the delegator, it receives is a DHKE key share and an enclave quote generated by the delegator. The PRE enclave passes the quote to the EPID verification service to validate the signature over the quote. If the verification succeeds, the PRE enclave sends its DHKE key share and enclave quote to the delegator, and waits for the delegator sending the decryption key in the next round's communication. The messages in remote attestation is output by `sgx_ra_get_msg1()` and `sgx_ra_proc_msg2()` provided by Intel SGX. Finally, when the proxy receives the encrypted decryption key of the delegator, it runs `sgx_ra_get_keys()` to get the negotiated key of this remote attestation, and decrypts in the enclave to get the decryption key from the delegator. The PRE enclave stores the decryption key in the SK Table in the trusted memory, and loads the delegator's encryption key in the PK Table. The proxy tags the key pair with the corresponding index. It then runs `sgx_ra_close()` to release the remote attestation context.

The encryption and re-encryption key generation done by the delegator are simple encryption and signature algorithms respectively. As for re-encryption, the delegator sends the encrypted query message and the signature. The query includes the metadata of the specific file waiting to be re-encrypted, along with the index of the target delegatee. The proxy decrypts it to get the message, and loads the corresponding encrypted data into the PRE enclave. Then it runs the ECALL function `Enclave_reencrypt()`.

There are three phases in the re-encryption step – decryption, re-encryption and signing. The PRE enclave first loads the delegator's decryption key from the SK Table. Then it runs `delegator_decrypt()` inside the enclave which decrypts the encrypted input data with the decryption key using the corresponding encryption scheme. As for re-encryption, the PRE enclave loads the encryption key of the delegatee from the PK Table, and then runs `delegatee_reencrypt()` which encrypts the decrypted data with the delegatee's encryption key to get the re-encrypted data. Subsequently, The PRE enclave signs the re-encrypted data with the ECDSA signing key which has already generated by `signkey_generate()`. Finally the PRE enclave outputs the re-encrypted data with the corresponding signature, and sends both of them to the target delegatee.

*2) Delegator Client:* Initialization of the delegator client is similar. The delegator first loads `Client.so`. The `MRENCLAVE` of the PRE enclave is statically defined in the enclave. When the delegator's client enclave is initialized, it runs the ECALL function `vkey_generate()` to generate an ECDSA key pair. Then it secretly keeps the signing key, and publishes the corresponding verification key. This key pair is used to sign the re-encryption message, which is regarded as the re-encryption key in `hPRESS`. In another case, when this client enclave restarts, it reloads the same key pair. Next, the delegator connects the proxy platform running the PRE enclave. It sends a DHKE key share and an enclave quote. The proxy responds with its DHKE key share and quote after verification. The delegator also verifies the response with the EPID verification service to identify that the message comes from the intended proxy platform. If the verification succeeds, the delegator runs `sgx_ra_get_keys()` to get the negotiated key, and encrypts its decryption key of the delegatee. The decryption key is sent to the proxy in the secure channel of Intel SGX.

When the delegator intends to start re-encryption, it generates a re-encryption message containing necessary information, and signs the message with the generated signing key. Then the delegator sends the re-encryption query message as well as the corresponding signature to the proxy.

### B. Performance Evaluation

To show that our `hPRESS` has better performance, we test the implementation of SGX instance of `hPRESS` on three different platforms. The first is a PC running an Intel (R) Core (TM) i7-7700 processor at 3.60 GHz with 16 GiB of RAM. The second is an Intel NUC running an Intel Core i3-6100U CPU 2.30GHz * 4. The third is another Intel NUC of Intel (R) Core (TM) i7-6770HQ CPU 2.60GHZ * 8 with 7.6 GiB of memory. All of their operating systems are Ubuntu 16.04. The code is developed and compiled using C++ with the Intel (R) SGX SDK 2.1 and Intel (R) SGX PSW 2.1 add-ons. The SGX applications are compiled with the 64-bit Debug mode build configuration[2].

As introduced, a standard PRE scheme consists of five basic phases: KEYGEN, ENC, REKEYGEN, REENC and DEC. ENC and DEC are same as those in the encryption schemes the delegator and the delegatee choose, which is independent of the performance evaluation of our proposal. Moreover, KEYGEN is a one-time cost in our design. Thus, we mainly focus on the performance of REKEYGEN and REENC. In `hPRESS`, KEYGEN is an ECDSA signing function in the delegator client which is used to sign the necessary information in the re-encryption query sent to the proxy, such as the metadata, the delegatee's tag. The signature is regarded as the re-encryption key. The API representing REENC in `hPRESS` is an ECALL function `Enclave_reencrypt()` in the PRE Enclave of the proxy platform.

Note that all of our results are tested locally, and we do not consider the network performance. The reason is that all the attestation protocols are implemented during the key generation and initialization, which is a one-time operation, thus it has little impact on the performance of the entire

[2]The reason why we do not adopt the Release Mode is that, currently to build enclave applications in the Release Mode requires a license from Intel, which is not necessary for academic researches.

TABLE I: Average Running Time of Each Step. (NUC_i3)

| Procedure | Time(ms) |
|---|---|
| Enclave creation | 64 |
| ECDSA setup | 8 |
| Message signing | 0.1 |
| Re-encrypting per block (Omit KeyGen) | 1.6 |

system. During re-encryption, network conditions have the same effects on hPRESS and other schemes.

Table I contains the running time of the steps in REKEYGEN and REENC. As discussed above, REKEYGEN includes message signing, and REENC includes enclave creation, re-encryption within the enclave and message signing. Although our performance test has a smaller scale than the data size of real-world scenario, even as the scale grows larger, the cost of message signing is still acceptable in practice.

To show that hPRESS has better speed and applicability, we also compare its performance with several previous cryptographic re-encryption schemes. We also test the performance of hPRESS under different settings where the delegator and delegatee adopt different kinds of encryption schemes.

First, we compare hPRESS with several previous cryptography-based broadcast proxy re-encryption schemes (BPRE). The comparison is meaningful because similar to hPRESS, BPRE intends to provide re-encryption services in 1-to-N cloud scenarios, and previous BPRE schemes are completely implemented with cryptography. To provide a performance comparison, we evaluate the execution time of the following five algorithms: Encrypting (Enc), Re-encryption key generating (ReKeyGen), Re-encrypting (ReEnc), Original ciphertext decrypting (Dec(Or)), and Re-encrypted ciphertext decrypting (Dec(Re)), and compare the results with other published BPRE schemes. Note that Enc and Dec(Or) are the same as the encryption and decryption algorithms chosen by the delegator, and Dec(Re) is the decryption algorithm of the delegatee's encryption scheme. Most cost of identity authentication lies in the key generation and initialization. ReKeyGen consists of generating a signature of the necessary information of the re-encryption queries, which costs little. Therefore, the performance of these phases in hPRESS could be improved significantly, and we make sufficient tests to prove it. We assume that the delegator aims to re-encrypt the message to 10 different delegatees, and the corresponding execution time is shown in Table II. We test the case where the delegator uses RSA with OAEP padding as the encryption scheme. The results show that compared to the cryptographic BPRE schemes, our hPRESS takes much less time, and we think the reason might be that cryptographic schemes spend more time on redundant operations such as modular multiplication and exponentiation. We utilize secure hardware to guarantee the data security in hPRESS, which achieves the same security level but costs less.

We test the performance of hPRESS in different settings where the delegator and delegatee change their encryption schemes, to find out the effects of the delegator and delegatee's encryption schemes to the execution time of the hPRESS. We assume that the performance of hPRESS is positively correlated to the performance of the encryption schemes of the delegator and delegatee, and in the ideal situation, it should be a linear relationship, because the proxy decrypts and encrypts in a separate manner. To prove our assumption, we utilize the BN library of OpenSSL to implement a simple ElGamal encryption algorithm with some latency. We assume that one of the delegator or delegatee uses this low-speed encryption scheme, and we test the corresponding performance. The results show that the performance of hPRESS is positively related to the performance of the delegator and delegatee's schemes as expected. Another factor that influences the performance of hPRESS is the performance of the proxy's hardware. We implement hPRESS on two different machines – NUC_i3, NUC_i7 and PC, and find that the CPU performance is dominant in hPRESS. It is predictable because hPRESS is more dependent on the SGX hardware, mainly including the enclave initialization and re-encryption within the PRE enclave. The platform with a more powerful CPU (i7) has a significant performance improvement than the i3-CPU. In comparison, the memory region size has less impact on performance. The reason is that the test data in the experiments is not in a large scale. Note that when the data size becomes larger than the enclave size, the performance slows down and a slicing strategy needs to be taken into consideration. On the contrary, traditional PRE schemes based on cryptographic algorithms are not that dependent on hardware conditions like hPRESS.

### C. Symmetric Encryption Extensions

Another test aims to integrate symmetric cryptography into hPRESS as expected. As mentioned in Section II, symmetric cryptography can be easily merged into hPRESS. If the delegator uses symmetric encryption scheme, the delegator sends the symmetric key to the proxy instead of the private key in the public-key schemes. The index of the sysmetric key is published instead of the actual encryption key, and other steps are the same as those in the public-key case. If the delegatee uses symmetric encryption schemes, the proxy has to get the encryption key of the delegatee to accomplish the re-encryption. However, in this scenario, the delegatee is also required to support Intel SGX to deliver the key to the proxy. It is not difficult because the key generation and initialization in hPRESS is the same to all the clients to catch up, despite it is the potential delegators and delegatees. Once the delegatee tries to catch up in hPRESS, the client immediately communicates with the proxy and sends the corresponding decryption key (no matter symmetric key or private key) to the PRE Enclave on the proxy platform. The decryption key will be stored securely by the proxy, so the delegator and delegatee using symmetric encryption schemes can also catch up in hPRESS smoothly.

We implement the above idea to see whether it works or not, and to evaluate the corresponding performance. We test the performance of hPRESS when one or both of the delegator and delegatee use a symmetric encryption scheme. The results are shown in Table III. The symmetric encryption scheme in the table is AES-GCM which is supported by EVP functions of OpenSSL, and the public-key encryption scheme

TABLE II: Average Execute Time of Each Phase of Different PRE Schemes (ms)

| PRE Scheme | Enc | Dec(Or) | ReKeygen | ReEnc | Dec(Re) | Total |
|---|---|---|---|---|---|---|
| [10] | 10.02 | 10.08 | 18.22 | 99.22 | 12.11 | 149.7 |
| [11] | 10.64 | 11.09 | 19.22 | 108.32 | 12.27 | 161.5 |
| [12] | 20.77 | 33.91 | 15.91 | 33.30 | 49.41 | 153.3 |
| [13] | 16.83 | 17.43 | 22.56 | 32.67 | 18.65 | 108.1 |
| hPRESS (NUC_i3) | 0.18 | 1.95 | 0.71 | 13.79 | 1.95 | 18.58 |
| hPRESS (NUC_i7) | 0.16 | 1.36 | 0.45 | 12.41 | 1.36 | 15.74 |
| hPRESS (PC) | 0.16 | 1.33 | 0.43 | 11.27 | 1.33 | 14.52 |

TABLE III: Time of ReEnc of hPRESS in Different Scenarios. (NUC_i3)

| Index | Delegator | Delegatee | Time of ReEnc |
|---|---|---|---|
| I | Public-key | Public-key | 13.79ms |
| II | Symmetric | Symmetric | 0.26ms |
| III | Public-key | Symmetric | 2.29ms |

is the mentioned RSA. The results show that when we involve symmetric cryptography in hPRESS, the performance is much better than the case where the delegator and the delegatee use public-key encryption schemes. The reason is that the performance of hPRESS mainly depends on and the performance of encryption schemes of the delegator and delegatee. It is obvious that when dealing with the data of the same size, most symmetric encryption schemes get a significant performance improvement compared with public-key on implementation.

## VII. Related Work

It has been over 20 years since its invention by Blaze et al. [14] in 1998.

In 1998, Blaze et al. [15] proposed the first PRE scheme, which is a bidirectional PRE scheme based on ElGamal. The researchers have made efforts to equip the PRE scheme with versatile capabilities [16]–[19], since PRE was born.

In 2005, Ateniese et al. [16] first proposed the construction of a unidirectional PRE scheme based on bilinear maps. Their scheme achieves to be non-transitive and collusion resistant, but it only offers chosen plaintext security which is weak for many practical applications. Their paper is also the first one that proposed the problem of constructing a multi-hop PRE scheme. Libert et al. [10] presented the first unidirectional proxy re-encryption schemes with chosen-ciphertext security in the standard model. Their system provably fits a unidirectional extension of the Canetti-Hohenberger security model [20]. Subsequently, Shao and Cao [18] proposed a unidirectional PRE scheme without pairing, however, their scheme was later proved to be vulnerable to CCA attack by Chow et al. in [19]. Moreover, they proposed another unidirectional PRE scheme that does not use bilinear pairing. Their scheme is CCA-secure and resists collusion attack. Moreover, most PRE schemes are single-hop. Only a few schemes beginning with [15] are multi-hop, but bidirectional. Green and Chu [21], [22] also provided other constructions of multi-hop PRE schemes, but the ciphertext size grows linearly with the number of re-encryptions. Subsequently, Liang et al. [23] proposed a multi-hop identity-based PRE (MH-IBPRE), which maintains the constant ciphertext size and computational complexity regardless of the number of re-encryption hops. Their scheme is proven secure, but still bidirectional.

To contain the two features together in a single scheme, Hohenberger et al. [24] first proposed constructions of multi-hop, unidirectional schemes based on program obfuscation, which was further researched in [25]. Inspired by this idea, Gentry [26] gave a generic construction of PRE from fully homomorphic encryption. They then extended their multi-hop unidirectional re-encryption constructions with succinct ciphertexts based on a standard assumption [27], which achieves CPA-secure. Phong et al. [28] proposed a CPA-secure PK-PRE scheme in the standard model which enjoys both unidirectionality and multi-hop delegation. Their scheme does not rely on bilinear maps, instead, the security of their construction is based on the hardness of the standard Learning-With-Errors (LWE) problem. Fan and Liu [2] proposed the first construction of a CCA-secure multi-hop, unidirectional PRE scheme based on lattices. In their proposal, they also rigorously defined the security models for the multi-hop setting.

Researchers have made great efforts to make PRE schemes with both high-level security and good features, and the latest PRE achieves unidirectional and multi-hop features while guaranteeing CCA-secure [2]. However, these PRE schemes still suffer from practical problems. An applicable PRE scheme requires not only the security and functional features, but also those practical ones. To the best of our knowledge, there is still a unidirectional multi-hop PRE scheme which achieves both constant decryption efficiency and no ciphertext extension.

There are also other researches of PRE aiming to construct PRE with other good features. In the early years, PRE was proposed based on the traditional public-key infrastructure setting which incurs complicated certificate management [29]. To make PRE more practical and secure, it needs to be combined with other cryptographic technologies. By combining PRE with identity based encryption (IBE), several identity-based PRE schemes (IPRE) were proposed [21], [22], [30]. In IPRE schemes, the receivers' recognizable identities can be regarded as public keys, and the delegator and the proxy are required only to know the delegatees' identities, which is more convenient.

However, the re-encryption processes in the early PRE schemes were executed in an all-or-nothing manner, which means that the proxy can either re-encrypt all the initial

ciphertexts or none of them. This control strategy limits the application of PRE systems. To improve the control strategy, a refined concept named as conditional PRE (CPRE) was proposed [30]–[33]. In CPRE strategy, only the ciphertexts with the specified condition are authorized for re-encryption by the proxy holding the corresponding re-encryption key. Another limitation of PRE and IPRE is that only a single delegatee is allowed. When there are multiple delegatees, PRE should be invoked several times. To satisfy the demand, broadcast PRE (BPRE) is proposed. In BPRE, a delegator is allowed to generate an initial ciphertext to a delegatee set, instead of a single delegatee. Chu et al. [31] presented a generalized notion of conditional proxy broadcast re-encryption (CPBRE). In their scheme, a delegator can delegate decryption rights to a set of delegatees at a time, and the scheme is secure against replayable chosen-ciphertext attacks (RCCA). In 2016, by incorporating CPRE, IPRE and BPRE, Xu et al. [12] proposed a versatile primitive referred to as conditional identity-based broadcast PRE (CIBPRE), and they formalized its semantic security. They implemented an application of the CIBPRE to a secure cloud email system. Their scheme made great progress to make PRE available for the cloud. However, there are still several limitations. In the real cloud environment, there are other factors besides identity that influence authorization, such as location and other information. Thus attribute-based encryption (ABE) was proposed based on IBE, which is more flexible and has been widely used in cloud [34]. Zhang et al. [35] proposed a novel technique called match-then-re-encrypt, which uses special components of the proxy re-encryption key and ciphertext to anonymously check whether the proxy can fulfill a proxy re-encryption or not.

Besides, other researches aim at optimizing PRE in other aspects. Hanaoka et al. [36] presented the first generic construction of a chosen-ciphertext secure unidirectional proxy re-encryption scheme. Isshikiet al. [37] presented a full CCA security definition to extend Hanaoka's scheme, and then proposed their improved PRE scheme. Kirshanova [38] presented a provably CCA-1 secure proxy re-encryption scheme in the selective model under the LWE assumption. Yang et al. [39] presented a ciphertext-policy attribute-based CPRE scheme, and provided a formalization of the primitive and its security proof. They further proposed to apply their scheme for fine-grained encryption of cloud data. Su et al. [40] proposed a multi-element access control condition based on the PRE scheme for the security requirement of mobile cloud computing. Their scheme achieved the target of multi-element PRE without increasing the amount of users' private keys in mobile cloud computing. Kim and Lee [41] tried to apply PRE in IoT devices. They proposed a method to implement PRE to manage data with fewer encryptions, and to provide a data sharing function to supplement the insufficient capacity of lightweight device networks. Moreover, Syalim et al. [3] extended symmetric cryptography to the concepts of PRE. They proposed an idea to implement a pure PRE for the symmetric ciphers by first transforming the plaintext into a random sequence of blocks using the all or nothing transform. Involving symmetric cryptography into PRE is a good try, although it did not completely address the limitation of existing PRE.

Although researches about PRE grow fast in the recent decades, there is still not a unidirectional multi-hop PRE with constant decryption efficiency and without ciphertext extensions. Besides, no existing PRE supports ciphertext transformation between arbitrary encryption scheme. Although PRE is potentially beneficial for many scenarios, it is still not full-fledged as expected without these features. Currently, PRE is only applied to a small range of its potential applications, such as key management [42] and relative applications [43]. Moreover, the performance of most of these existing schemes still too weak to for practice, which makes them harder to be applied on other real-world applications.

## VIII. Conclusion

Proxy re-encryption is a powerful cryptographic mechanism. A delegator can deliver the proxy a re-encryption key to re-encrypt the initial ciphertext to the delegatee's ciphertext. There are many potential scenarios of PRE, such as digital right management and online car rental. However, existing PRE schemes are not practical enough to be adopted in real-world applications of these scenarios. The main reason is that existing PRE schemes cannot cover all necessary features for practice. Most existing PRE schemes have weak performance and scalability, which narrows the scalability of PRE and makes it hard to adapt smoothly into existing cryptographic systems.

In this paper, we extend the concept of PRE, provide detailed definitions and security analysis, and propose a hardware-enhanced PRE scheme with secure enclave, called `hPRESS`. To the best of our knowledge, `hPRESS` is the first unidirectional multi-hop PRE scheme with constant decryption efficiency and constant ciphertext size without extensions. Moreover, `hPRESS` greatly improves the weak performance of PRE with secure hardware, and achieves better scalability which supports ciphertext transformations between arbitrary encryption schemes, including public-key encryption and symmetric key encryption. We present a detailed description of our proposal, and provide a security analysis of the `hPRESS`. Finally, we discuss the implementation details of an instance and present a performance evaluation. Our results prove that the `hPRESS` keep the CCA security of `hPRESS`with Intel SGX. The results show that, `hPRESS` has better features and performance than existing algorithm-based PRE schemes, which we think is a significant improvement of PRE researches.

## Acknowledgments

This work was supported in part by Alibaba-Zhejiang University Joint Institute of Frontier Technologies, by Zhejiang Key R&D Plan (Grant No. 2019C03133), by Major Scientific Research Project of Zhejiang Lab (Grant No. 2018FD0ZX01), by Young Elite Scientists Sponsorship Program by CAST (Grant No. 17-JCJQ-QT-045), by the Fundamental Research Funds for the Central Universities (Grant No. 2020QNA5021), by National Natural Science Foundation of China (Grant No. 61772236 and Grant No. 61972094), by Leading Innovative and Entrepreneur Team Introduction Program of Zhejiang

(Grant No. 2018R01005), and by Research Institute of Cyberspace Governance in Zhejiang University. Cong Zuo has been supported by the CSIRO Data61 scholarship.

## REFERENCES


[1] G. Taban, A. A. Cardenas, and V. D. Gligor, "Towards a secure and interoperable drm architecture," pp. 69–78, 2006.

[2] X. Fan and F. Liu, "Proxy re-encryption and re-signatures from lattices," pp. 363–382, 2019.

[3] A. Syalim, T. Nishide, and K. Sakurai, "Realizing proxy re-encryption in the symmetric world," in *International Conference on Informatics Engineering and Information Science*. Springer, 2011, pp. 259–274.

[4] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 991–1008.

[5] Y. Xiao, M. Li, S. Chen, and Y. Zhang, "Stacco: Differentially analyzing side-channel traces for detecting SSL/TLS vulnerabilities in secure enclaves," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 859–874.

[6] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: functional encryption using Intel SGX," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 765–782.

[7] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel® software guard extensions: Epid provisioning and attestation services," *White Paper*, vol. 1, pp. 1–10, 2016.

[8] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee, "Obliviate: A data oblivious file system for Intel SGX," in *25th Annual Network and Distributed System Security Symposium, NDSS*, 2018, pp. 18–21.

[9] M.-W. Shih, S. Lee, T. Kim, and M. Peinado, "T-SGX: Eradicating controlled-channel attacks against enclave programs." in *NDSS*, 2017.

[10] B. Libert and D. Vergnaud, "Unidirectional chosen-ciphertext secure proxy re-encryption," in *International Workshop on Public Key Cryptography*. Springer, 2008, pp. 360–379.

[11] J. Weng, M. Chen, Y. Yang, R. Deng, K. Chen, and F. Bao, "CCA-secure unidirectional proxy re-encryption in the adaptive corruption model without random oracles," *Science China Information Sciences*, vol. 53, no. 3, pp. 593–606, 2010.

[12] P. Xu, T. Jiao, Q. Wu, W. Wang, and H. Jin, "Conditional identity-based broadcast proxy re-encryption and its application to cloud email," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 66–79, 2016.

[13] M. Sun, C. Ge, L. Fang, and J. Wang, "A proxy broadcast re-encryption for cloud data sharing," *Multimedia Tools and Applications*, vol. 77, no. 9, pp. 10 455–10 469, 2018.

[14] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, 1976.

[15] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 127–144.

[16] G. Ateniese, K. Fu, M. Green, and S. Hohenberger, "Improved proxy re-encryption schemes with applications to secure distributed storage," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 1, pp. 1–30, 2006.

[17] R. Canetti and S. Hohenberger, "Chosen-ciphertext secure proxy re-encryption," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 185–194.

[18] J. Shao and Z. Cao, "CCA-secure proxy re-encryption without pairings," in *International Workshop on Public Key Cryptography*. Springer, 2009, pp. 357–376.

[19] S. S. Chow, J. Weng, Y. Yang, and R. H. Deng, "Efficient unidirectional proxy re-encryption," in *International Conference on Cryptology in Africa*. Springer, 2010, pp. 316–332.

[20] R. Canetti, H. Krawczyk, and J. B. Nielsen, "Relaxing chosen-ciphertext security," pp. 565–582, 2003.

[21] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in *International Conference on Applied Cryptography and Network Security*. Springer, 2007, pp. 288–306.

[22] C.-K. Chu and W.-G. Tzeng, "Identity-based proxy re-encryption without random oracles," in *International Conference on Information Security*. Springer, 2007, pp. 189–202.

[23] K. Liang, C. Chu, X. Tan, D. S. Wong, C. Tang, and J. Zhou, "Chosen-ciphertext secure multi-hop identity-based conditional proxy re-encryption with constant-size ciphertexts," *Theoretical Computer Science*, vol. 539, pp. 87–105, 2014.

[24] S. Hohenberger, G. N. Rothblum, A. Shelat, and V. Vaikuntanathan, "Securely obfuscating re-encryption," pp. 233–252, 2007.

[25] N. Chandran, M. Chase, and V. Vaikuntanathan, "Functional re-encryption and collusion-resistant obfuscation," vol. 7194, pp. 404–421, 2012.

[26] C. Gentry, "Fully homomorphic encryption using ideal lattices," pp. 169–178, 2009.

[27] N. Chandran, M. Chase, F. Liu, R. Nishimaki, and K. Xagawa, "Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices," vol. 8383, pp. 95–112, 2014.

[28] L. T. Phong, L. Wang, Y. Aono, M. H. Nguyen, and X. Boyen, "Proxy re-encryption schemes with key privacy from lwe." *IACR Cryptology ePrint Archive*, vol. 2016, p. 327, 2016.

[29] A. Boldyreva, M. Fischlin, A. Palacio, and B. Warinschi, "A closer look at pki: Security and efficiency," in *International Workshop on Public Key Cryptography*. Springer, 2007, pp. 458–475.

[30] J. Shao, G. Wei, Y. Ling, and M. Xie, "Identity-based conditional proxy re-encryption," in *2011 IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–5.

[31] C.-K. Chu, J. Weng, S. S. Chow, J. Zhou, and R. H. Deng, "Conditional proxy broadcast re-encryption," in *Australasian Conference on Information Security and Privacy*. Springer, 2009, pp. 327–342.

[32] J. Weng, Y. Yang, Q. Tang, R. H. Deng, and F. Bao, "Efficient conditional proxy re-encryption with chosen-ciphertext security," in *International Conference on Information Security*. Springer, 2009, pp. 151–166.

[33] L. Fang, W. Susilo, and J. Wang, "Anonymous conditional proxy re-encryption without random oracle," in *International Conference on Provable Security*. Springer, 2009, pp. 47–60.

[34] J. Li, W. Yao, Y. Zhang, H. Qian, and J. Han, "Flexible and fine-grained attribute-based data storage in cloud computing," *IEEE Transactions on Services Computing*, vol. 10, no. 5, pp. 785–796, 2017.

[35] Y. Zhang, J. Li, X. Chen, and H. Li, "Anonymous attribute-based proxy re-encryption for access control in cloud computing," *Security and Communication Networks*, vol. 9, no. 14, pp. 2397–2411, 2016.

[36] G. Hanaoka, Y. Kawai, N. Kunihiro, T. Matsuda, J. Weng, R. Zhang, and Y. Zhao, "Generic construction of chosen ciphertext secure proxy re-encryption," in *Cryptographers' Track at the RSA Conference*. Springer, 2012, pp. 349–364.

[37] T. Isshiki, M. H. Nguyen, and K. Tanaka, "Proxy re-encryption in a stronger security model extended from ct-rsa2012," in *Cryptographers' Track at the RSA Conference*. Springer, 2013, pp. 277–292.

[38] E. Kirshanova, "Proxy re-encryption from lattices," in *International Workshop on Public Key Cryptography*. Springer, 2014, pp. 77–94.

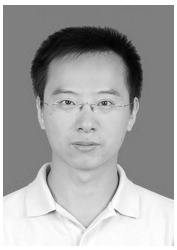
[39] Y. Yang, H. Lu, J. Weng, Y. Zhang, and K. Sakurai, "Fine-grained conditional proxy re-encryption and application," in *Provable Security*, S. S. M. Chow, J. K. Liu, L. C. K. Hui, and S. M. Yiu, Eds. Cham: Springer International Publishing, 2014, pp. 206–222.

[40] M. SU, G. Shi, R. Xie, and A.-m. Fu, "Multi-element based on proxy re-encryption scheme for mobile cloud computing," *Journal on Communications*, vol. 36, no. 11, pp. 73–79, 2015.

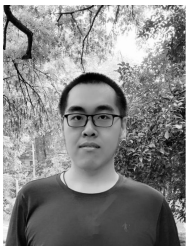
[41] S. Kim and I. Lee, "IoT device security based on proxy re-encryption," *Journal of Ambient Intelligence and Humanized Computing*, vol. 9, no. 4, pp. 1267–1273, 2018.

[42] M. Egorov and M. Wilkison, "Nucypher kms: Decentralized key management system." *arXiv: Cryptography and Security*, 2017.

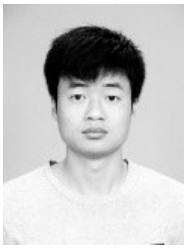
[43] CarBlock, "A global transportation data protocol with decentralized applications," https://www.carblock.io/.



**Fan Zhang** received the Ph.D. degree from the Department of Computer Science and Engineering, University of Connecticut, in 2012. He is currently an Associate Professor with the College of Computer Science and Technology, Zhejiang University. His research interests include system security, hardware security, cryptography, and computer architecture.



**Ziyuan Liang** is currently working towards the Ph.D. degree in Information Science & Electronic Engineering, and also with the School of Cyberspace Research, Zhejiang University, Hangzhou, China. His research interests include secure computing, Intel SGX technique and side channel attacks.



**Cong Zuo** received a B.S. degree from the School of Computer Engineering, Nanjing Institute of Technology, and an M.S. degree from the School of Computer Science and Information Engineering, Zhejiang Gongshang University, China. He is currently pursuing a Ph.D. degree with Monash University under the supervision of Dr. Joseph K. Liu and Shi-Feng Sun. He is also affiliated with Data61, and his Data61 supervisor is Josef Pieprzyk. His main research interest is applied cryptography.

**Jun Shao** received the Ph.D. degree from the Department of Computer Science and Engineering at Shanghai Jiao Tong University, Shanghai, China in 2008. He was a postdoc in the School of Information Sciences and Technology at Pennsylvania State University, USA from 2008 to 2010. He is currently a professor of the School of Computer and Information Engineering at Zhejiang Gongshang University, Hangzhou, China. His research interests include network security and applied cryptography.

**Jianting Ning** received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University in 2016. He is currently a Professor with the Fujian Provincial Key Laboratory of Network Security and Cryptology, School of Mathematics and Computer Science, Fujian Normal University, China. He is also a research fellow at School of Information Systems, Singapore Management University. His research interests include applied cryptography and information security.

**Jun Sun** is currently an associate professor at School of Information Systems, Singapore Management University (SMU). He received his Ph.D. in computing science from National University of Singapore (NUS) in 2006. In 2007, he received the prestigious LEE KUAN YEW postdoctoral fellowship. He has been a faculty member in SUTD since 2010. He was a visiting scholar at MIT from 2011-2012. Jun's research interests include software engineering, formal methods, program analysis and cyber-security.

**Joseph Liu** is an Associate Professor in the Faculty of Information Technology, Monash University. He got his PhD from the Chinese University of Hong Kong at 2004. Prior to joining Monash at 2015, he has worked as a research scientist at Institute for Infocomm Research (I2R) in Singapore for more than 7 years. His research areas include cyber security, blockchain, IoT security, applied cryptography and privacy enhanced technology. He has received more than 7500 citations and his H-index is 50, with more than 200 publications in top venues such as CRYPTO, ACM CCS. He is currently the lead of the Monash Cyber Security Discipline Group. He has established the Monash Blockchain Technology Centre at 2019 and serves as the founding director. He has been given the Dean's Award for Excellence in Research Impact in 2018, and the prestigious ICT Researcher of the Year 2018 Award by the Australian Computer Society (ACS), the largest professional body in Australia representing the ICT sector, for his contribution to the blockchain and cyber security community.

**Yibao Bao** is a senior security expert of the fundamental technology research team, security department of Alibaba Group, is engaged in the research field of cryptography and data security. Currently, he is mainly engaged in hardware cryptography and Intel SGX security, focusing on its application in high-performance cryptography, intellectual property protection, advanced cryptography application, and is committed to applying it to cloud computing and other distributed computing environments.