Reducing HSM Reliance in Payments through Proxy Re-Encryption[*][†]

Sivanarayana Gaddam
Visa

Atul Luykx
Security Engineering Research, Google

Rohit Sinha
Swirlds Inc.

Gaven Watson
Visa Research

Abstract

Credit and debit-card payments are typically authenticated with PINs. Once entered into a terminal, the PIN is sent as an encrypted *PIN block* across a payments network to the destination bank, which decrypts and verifies the PIN block. Each node in the payments network routes the PIN block to the next node by decrypting the block with its own key, and then re-encrypting the PIN block with the next node's key; nodes establish shared secret keys with their neighbors to do so. This decrypt-then-encrypt operation over PIN blocks is known as *PIN translation*, and it is currently performed in Hardware Security Modules (HSMs) to avoid possible PIN exposure. However, HSMs incur heavy acquisition and operational expenses.

Introduced at EUROCRYPT'98, proxy re-encryption (PRE) is a cryptographic primitive which can re-encrypt without exposing sensitive data. We perform an extensive study of PRE as applied to PIN translation, and show through formalization, security analysis, and an implementation study that PRE is a practical alternative to HSMs. With PRE, we eliminate the need for HSMs during re-encryption of a PIN, thus greatly reducing the number of HSMs needed by each participant in the payments ecosystem. Along the way we conduct practice-oriented PRE research, with novel theoretical contributions to resolve issues in comparing so-called honest re-encryption to chosen-ciphertext PRE security, and a new efficient PRE scheme achieving a type of chosen-ciphertext security.

1 Introduction

Credit and debit-card fraud is a multi-billion dollar problem. To help combat it, financial institutions commonly require consumers to enter a PIN during payment to further verify the transaction's authenticity. Traditionally, PIN-based verification was reserved for use at ATMs, but with the adoption of EMV [40] (a.k.a. Chip-and-PIN), PINs are now used to authenticate both ATM and Point-of-Sale (PoS) transactions.

PIN verification on a PoS device is either *offline* or *online*. Offline PIN verification only involves the PoS device and the chip card. The consumer-entered PIN is sent directly from the PIN pad to the chip card, where it is authenticated.

In contrast, online PIN verification involves sending the PIN to the consumer bank that issued the card. The PoS device transmits the PIN in encrypted form — along with other sensitive data, such as personal account numbers (PANs) — across a network to the consumer's bank, which verifies the PIN before authorizing the transaction. The PIN might pass through many intermediaries over the network, including PoS terminals, payment gateways, merchant banks, network processors and consumer banks. Countries with mature payment infrastructure perform nearly all PIN verifications online. For example, in the USA offline PIN verification has not been adopted due to the complexity involved in synchronizing the offline PIN held on the chip card with the backend PIN used for online verification [71].

Many works and consumer reports discussing the security of PINs focus on attacks at the point of entry. Divulging a PIN to an attacker that can steal or clone cards could lead to immediate loss of funds. ATM skimming and overlay attacks continue to be a major cause of fraud [69]. Of equal importance however, is to ensure that PINs remain secure as they are transmitted to the consumer bank.

Payments Card Industry (PCI) regulations [63] mandate that PINs should never be exposed during transmission. Naïvely, one might assume that the PoS terminal encrypts the PIN under a key shared with the consumer bank. However, as the PoS Partner has no direct relationship with the consumer bank, it is difficult for such a key to be established. Therefore, the payments ecosystem operates by having each pair of neighboring network participants separately establish a shared key; for example, the PoS Partner shares a key with the Payment Gateway, who in turn shares a different key with the Merchant Bank and so on (cf. Figure 1).

Currently, network participants comply with PCI regula-

[*] An abridged version of this work appears at Usenix Security 2021. This is the full version.

[†] Work conducted while all authors were employees of Visa Research.

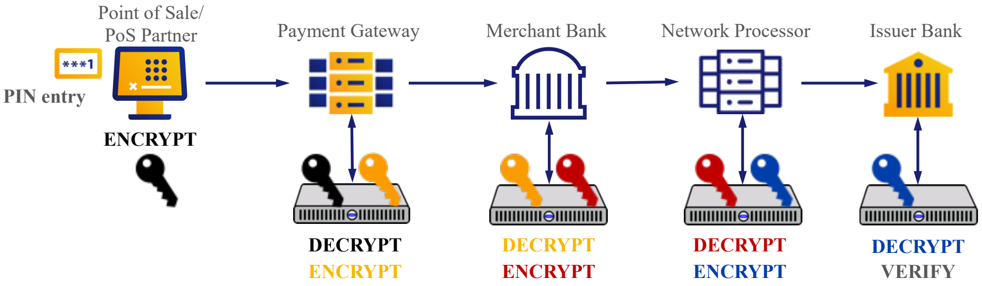

Figure 1: HSM Key setup and typical transaction flow for 3DES "decrypt-then-encrypt" PIN translation.

tions by using hardware security modules (HSMs) to decrypt-then-encrypt the PIN before forwarding to the next party. As a result, HSMs proliferate across the payments ecosystem starting from payment gateways. HSMs incur significant purchase and maintenance costs, with a single payment HSM typically costing around $10-20 000 before considering additional annual licensing and maintenance fees. Further costs result from the strict and specialized processes of operating HSMs. For example, firmware updates may require two collaborating administrators to apply the update on each individual appliance. PCI requires this dual-control operation to ensure that no one individual can compromise the protected keys, but the administration cost increases significantly as a result.

Moreover, HSMs become a performance bottleneck as the maximum throughput of payment transactions at any party depends on the number of available HSMs at that party, creating further issues when traffic spikes, such as during holidays. Unlike modern VM-based architectures (such as cloud services) which spin-up additional instances on-the-fly, HSM-based solutions lack this elasticity and must have additional HSMs deployment in advance. For example, network processors such as Visa process on average 5 000 transactions per second, and are designed to handle upwards of 60 000 transactions per second [6]; therefore, a large number of HSMs must be deployed to ensure sufficient hardware parallelism to handle this throughput.

Ideally, the payments ecosystem should not need to use HSMs to maintain security, and would operate with commodity hardware instead. However there are significant challenges to removing HSMs from the payments ecosystem. First and foremost, changing transaction processing is both time-intensive and expensive — for instance, it took nearly a decade to deploy EMV in the USA [4]. Any solution that replaces HSMs should preferably be backwards compatible and easy to deploy. Second, due to the large number of parties in the payments ecosystem, each with their own priorities, the design space is large, yet a proper solution requires taking into account technical, regulatory, and business constraints.

Contributions

Despite the volume and importance of PIN-based transactions, secure PIN translation has received little attention from the research community. To address this we initiate a systematic study of PIN translation with an overarching aim to reduce HSM reliance. After detailing the current setting in Section 2, we distill a threat model and design goals for an improved approach in Section 3, with three main goals:

1. reducing HSM numbers,
2. ensuring key management is in-line with existing practices such as PCI [63], and
3. ensuring maximal backward compatibility.

Our threat model and design goals let us discuss seemingly straightforward solutions — such as simply deploying public-key encryption to transport PINs — and why they are unsuitable (cf. Section 3.3).

Based on this groundwork we provide the first mathematical formalization of secure PIN translation (cf. Section 4 and 5). We capture the actors, threat model and security goals of PIN authentication within the framework of provable security.

Our models allow us to rigorously explain how we can achieve our goals using Proxy Re-Encryption (PRE), a type of public-key encryption introduced by Blaze et al. [23] where ciphertext can be re-encrypted to other public keys without decrypting. In our search for a suitable PRE scheme, we find that existing schemes are either not secure enough, too inefficient, or are too restrictive in functionality (cf. Appendix C). As a result, we introduce modeling techniques and a new scheme, opening a new line of practice-oriented PRE research.

More specifically, our observations lead us to define new PRE notions which more accurately capture the abilities of an attacker in the payment setting. With chosen-ciphertext secure (CCA) schemes too inefficient and chosen-plaintext security (CPA) not enough, we seek a strengthening of security under Honest Re-encryption Attacks (HRA) [34] where adversaries may perform chosen-ciphertext attacks. However, as Cohen points out [34], CCA and HRA security are incomparable — neither implies the other — presenting a theoretical block to finding a good security definition. We circumvent the block by observing that many issues disappear when focusing on deterministic re-encryptions (as is relevant to our setting). We introduce a new PRE security definition, hrCCA, which is a hybrid of HRA and CCA security and links the two notions.

Furthermore, we introduce a new attack notion for PRE we call Ciphertext-Verification Attack (CVA). While weaker than CCA, this notion more closely aligns with the information available from a payment transaction. Consider PIN verification: the decrypted PIN is never returned but instead a single bit which signifies whether verification was successful or not is returned.

In Section 6.1, we present our efficient PRE scheme which follows the KEM-DEM[1] hybrid-encryption paradigm. We analyze the hrCCA security of this construction in Section 6.2.

Finally our experimental evaluation in Section 7 shows the throughput and latency metrics of our new construction on commodity machines. We demonstrate improved latency requirements at all intermediaries between the PoS device and the consumer bank, and show 5x improvement in throughput when using a single commodity server compared to a single HSM — we achieve over 6000 transactions per second using a single server, which can handle the average throughput faced by network processors such as Visa.

Our proposal can be deployed on the existing EMV [40] and ISO PIN [10] standards, without having to issue new cards to the customers or incur significant changes at the intermediaries. Additionally, all key-management remains inline with existing PCI-compliant processes. HSMs are still required for key ceremonies during the initial setup phase. However, after the key ceremony, the initial public key and subsequent re-encryption keys can be used without the need for trusted hardware. This greatly reduces the number of HSMs needed by the payment gateway, the merchant bank and other parties since they no longer need HSM infrastructure to cope with the scale of online processing of PIN translations.

2 An Overview of PIN-based Payments

2.1 Online PIN Verification

When PIN verification is online, the PIN passes through several intermediaries before it reaches the appropriate consumer bank. For simplicity, we focus on five participants — the PoS terminal and partner, payment gateway, merchant bank, network processor, and the consumer bank — although our results are more generally applicable.

Merchants employ PoS partners and payment gateways to accept payments from cardholders across the globe. PoS partners help merchants manage heterogenous PoS terminals manufactured by numerous vendors. Payment gateways help merchants accept payments from cardholders by contacting the merchant's bank. Merchant banks help merchants accept various card types by contacting the appropriate network processor (e.g. Visa). The network processor performs authorization by contacting the consumer bank, who performs the final verification of the transaction.

2.2 Key Management and Compliance

PCI regulations require that PINs are never in the clear, except inside secure hardware [63, Req 1]. Thus intermediaries must deploy HSMs to handle the PINs as they traverse the network. Keys are generated inside the HSMs and then shared with the neighboring party in a secure key ceremony process (Section 3.4) [63, Req 8] — such ceremonies are generally time consuming and require strict processes to be followed, cf. Section 3.4.1. Subsequently, when an encrypted PIN passes through an intermediary, it will be sent to the intermediary's HSM to be decrypted and then re-encrypted under the shared key of the next hop (cf. Figure 1).[2]

2.3 Encrypted PIN Blocks

PINs are transmitted over the payments network in the form of *Encrypted PIN Blocks* (EPBs). An EPB is constructed by first encoding the PIN and then encrypting this encoded block. PIN block encoding can be performed in multiple ways as defined in the ISO 9654-1 standard [10], named ISO Format 0 through 4. Format 2 is used for PINs that are submitted from the IC card reader to the IC card, and Format 1 is used when the PAN is not available, which in most payments cases is not the case. Hence, we focus on Formats 0 and 3 as these are relevant for our setting (cf. Figure 2. ISO Format 4 is the most recently defined, but is yet to see wide deployment. Formats 0 and 3 take as input the PIN and the card number (PAN) and output an encoded block. The main difference between these two formats is that Format 0 uses a fixed padding while Format 3 uses a random padding.

After encoding, the PIN block is encrypted using a block cipher to give the final EPB (cf. Figure 3). Formats 0 and 3 both output a 64-bit block which would then be encrypted using 3DES. 3DES remains the predominant block cipher used by the financial industry due to the difficulties presented by migrating large amounts of legacy hardware to new methods. ISO Format 4 defines a 128-bit encoded block which would instead be encrypted by AES. As mentioned previously, this is yet to see wide deployment.

3 Threat Model and System Requirements

3.1 Threat Model

Due to the sensitive nature of PINs it is important to ensure their security at all points across a payment transaction. Bank

[1] Key-Encapsulation Mechanism and Data-Encryption Mechanism — The KEM is a public-key encryption scheme which encrypts an ephemeral symmetric key used by the DEM to encrypt the data.

[2] Note that when going from PoS terminal to PoS Partner PCI requires that each transaction uses a unique key e.g. using the DUKPT method [13].

Figure 2: ISO PIN-Block Encodings (as defined in ISO 9564-1)

A PIN-block encoding scheme $\mathsf{ES} = (\mathsf{encode}, \mathsf{decode})$ take two inputs a PIN and a PAN, and outputs a PIN Block (PB). The encoded PB is the XOR of two fields, a plaintext PIN field and an account number field.

Plaintext PIN field:

C	N	P	P	P	P	P/F	P/F	P/F	P/F	P/F	P/F	P/F	P/F	F	F

where each 4-bit field is as follows:

- C – Control field: takes value 0000 (zero) for ISO-0 or 0011 (3) for ISO-3;
- N – PIN length: takes a value from 0100 (4) to 1100 (12);
- P – PIN digit: takes a values from 0000 (zero) to 1001 (9);
- F – Fill digit: takes value 1111 (15) for ISO-0 or a random value between 1010 (10) and 1111 (15) for ISO-3.

Account Number field:

0	0	0	0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	A_{11}	A_{12}

where each 4-bit field is as follows:

- 0 – Pad digit: takes value 0000 (zero);
- A_i – the 12 rightmost digits of the PAN excluding the check digit (values from 0000 (0) to 1001 (9)).

Figure 3: PIN-Block Encryption

```
function $\mathsf{SE.KG}(1^\lambda)$                         ▷ Key Generation
    $K \xleftarrow{\$} \{0,1\}^k$
    return $K$
function $\mathsf{SE.E}(K, PAN, PIN)$                       ▷ Encryption
    $PB \leftarrow \mathsf{encode}(PAN, PIN)$
    $EPB \leftarrow E(K, PB)$
    return $EPB$
function $\mathsf{SE.D}(K, PAN, EPB)$                       ▷ Decryption
    $PB \leftarrow D(K, EPB)$
    $PIN \leftarrow \mathsf{decode}(PAN, PB)$
    return $PIN$
```

customers are already well aware of the importance of not revealing their PIN to others but they are not the only party that must be concerned with PIN privacy. After entry at the PoS any entity which the PIN passes through must also ensure its security. As a result PCI specifies detailed security requirements that all entities handling PIN data must meet.

In line with PCI requirements we formally study PIN's security during their transmission from PoS to consumer bank. Specifically, we assume adversaries can compromise any intermediary or network link. Adversaries are permitted to drop and inject arbitrary messages, and tamper with any message between any two parties. They may corrupt any subset of parties to obtain their secret keys, such as the HSM-protected keys established during setup. However, security for the other parties should remain despite these corruptions.

To formally capture these capabilities, we enhance security notions for Proxy Re-encryption. In Section 4 we define security for Proxy Re-encryption, and discuss the unique setting of payments where the adversary has access to a restricted decryption oracle, which reveals a single bit of information whether each transaction's ciphertext encodes a valid PIN. Hence, we introduce a new notion for PRE under Ciphertext-Verification Attack (CVA), and justify in Section 5 why this notion captures security for the payments setting.

While the primary purpose of PINs is for authentication, we however focus on ensuring the privacy of PINs as they traverse the network, for the following reason. Due to the limited message space of PINs (10^4), an attacker's best strategy would always be a simple brute-force, entering each PIN in turn at the PoS device. However, an issuer or network processor can mitigate this attack by restricting the number of retries permitted through rate limiting. Instead, a more scalable and clandestine attack to break PIN confidentiality could be launched by a network attacker, or attackers who compromises a subset of the parties in the payments ecosystem. We analyze security with respect to such adversaries in this paper.

3.2 Design Goals

The primary security goal of any payment system handling PIN data is to ensure appropriate levels of privacy. Three additional goals we wish to achieve are to:

- reduce HSM numbers,
- ensure Key Management is in-line with existing practices, and
- ensure maximal backward compatibility.

Reducing HSM numbers. As PCI requirements mandate that PINs should only ever appear in the clear inside an HSM, it is not possible to completely eliminate HSMs from a payment transaction: the final entity performing PIN verification (i.e., the consumer bank) would still require an HSM to decrypt and verify the PIN. Nevertheless, we can eliminate HSM use during transaction processing by all other parties.

One caveat is that all parties must retain a small number of HSMs for key management. As with PINs, PCI mandates that secret keys must only be handled within a secure environment such as an HSM. As Key Management tasks are only performed a small number of times per year and can largely be performed offline, our overarching goal of reducing HSM numbers is still obtainable. This is due to corresponding parties no longer requiring a large farm of HSMs to cope with the huge performance demands of processing peak transaction volumes. The HSM's purpose now is purely for a small number of secure key-management operations. We explore the specifics of these operations next.

Key Management. Any new key-management operations we define must meet two important criteria. First as mentioned above, they must adequately meet or improve upon the requirements mandated by PCI [63]. Secondly, as parties within the network can only communicate with their immediate neighbors we must ensure that any key ceremony we define considers this fact. For example, it is not possible for a consumer bank to generate a key which would be injected into the PoS terminals. We discuss in detail the key management implications of our solution in Section 3.4.

Backward compatibility. The update or migration of any large-scale system is always a challenging task. Within the financial industry this is no different with many banks still operating mainframes and relying on 3DES for many crypto functions. Our solution must provide a straightforward deployment and migration path. It should also largely remain backwards compatible with existing specifications and require little changes to deployed hardware, that is, it should not require changes to existing PoS devices.

A final requirement to ensure backwards compatibility relates to the ISO 8583 [49] message format. As PIN blocks are passed between the different payment parties they are included within an ISO 8583 message. The message format which this defines restricts the space we have for any additional data. A number of data fields reserved for private use limit us to a few hundred bytes [66] which we may use for our purposes.

3.3 Failed Approaches

We discuss why traditional public-key cryptography based solutions fail to achieve seamless deployment. We focus on three salient solutions and highlight how they incur significant deployment overheads.

Alternative I. A naïve way to achieve PIN confidentiality were if the Point-of-Sale (PoS) device encrypted the PIN under the consumer's bank's public key. In this case, all parties involved in a PIN-based transaction could simply forward the encrypted PIN to the consumer bank. However, for this to work a PoS device must have secure access to a database which maps PANs to the corresponding consumer bank's public keys. Storing such a database on PoS devices is not feasible because it incurs significant deployment overheads (due to the large number of existing PoS devices — there are currently 46 million merchants accepting Visa cards [6]). Furthermore, such a database would be constantly evolving and hence a remote database which the PoS device could invoke to fetch the consumer bank's public key may seem more appropriate. However, maintaining such a central database across network processors would necessitate new interoperability requirements. Additionally the remote invocation would not only increase transaction latency, but also requires significant changes at various intermediaries (PoS partners, gateways, merchant banks, and network processors) as the transaction processing flow must be modified.
Main Concerns: deployment cost, backwards compatibility

Alternative II. Offline PIN verification [1] uses public keys stored on the chip card to encrypt PINs. Taking a similar approach for online PIN, where a consumer bank's public key stored on the card is used to encrypt the PIN seems an obvious solution. However, offline PIN verification is optional and no longer supported by banks in the USA [5, 9], therefore key material for this purpose is either not available or is not supported by the standard. On a similar note, we could consider leveraging the authorization key on the card (a symmetric key shared with the consumer's bank) for PIN encryption. This approach however is non-standard and requires new logic that necessitates the re-issuance of all chip cards, which is prohibitively expensive. It is also backward incompatible because not all merchants support chip cards [8], including many gas stations in the USA [3].
Main Concerns: deployment cost, backwards compatibility

Alternative III. The PIN could also be encrypted against a network processor's public key, but this seemingly simple solution is not deployable as merchants will lose *routing flexibility* of PIN-based transactions. Merchants today dynamically route PIN-based transactions [2] over their preferred network processors to minimize the processing fees. While one could encrypt against multiple network processors for each transaction, there are over 22 network processors globally, this would be both expensive for PoS devices and impossible to fit these ciphertexts within the available space allowed by the ISO 8583 standard [66]. To that end, the above seemingly natural solutions, including the solution proposed by Jayasinghe et al. [51], incurs significant changes to the ecosystem. Moreover, this approach requires the network processor (which is already the bottleneck for transaction processing) to perform public-key decryption for each transaction, incurring significant latency overheads.
Main Concerns: backwards compatibility, restricts routing flexibility

3.4 Key Management in Practice

3.4.1 Traditional Approach

In the traditional setting each pair of neighboring parties must establish a shared symmetric key. As all keys are held within HSMs, a key ceremony is strictly followed to establish these shared keys. Note that there is no connection between neighboring parties' HSMs which would facilitate an online key exchange. Instead each party designates several employees as *key custodians*. Two parties establish a shared key in the following way. For the purposes of our description each party has two custodians.

First, one of the parties (party A) generates a Key-Encryption Key (KEK) in their HSM. The two custodians at party A perform an export operation on the HSM such that each custodian receives an XOR-share of the key. The first custodian seals their XOR share in a tamper-proof envelope and sends this to the corresponding custodian at party B.

Similarly, the second custodian sends their XOR-share via a different courier to the corresponding custodian at party B. The custodians at party B then engage in an import operation on their HSM. Note that at no time has any one party had knowledge of the full plaintext key. This concept is known as *split knowledge*.

Once both parties have the same KEK loaded in their HSM, all subsequent keys can be shared encrypted under this key (where key generation and encryption under the KEK is performed inside the HSM). This is a strictly audited process and must closely follow the requirements mandated by PCI. In particular, the process described above closely follows Control Objective 3, Requirement 8 of the PCI PIN Security Requirements [63].

3.4.2 Key Management Requirements

Our goal is to design a scheme where the re-encryption key can be used in unprotected memory without fear of PIN exposure, i.e. knowledge of a re-encryption key should not enable an adversary to determine a customer's PIN. However, to establish these special re-encryption keys there still needs to be some form of trusted key ceremony. As a result we must assume that parties still use a minimal number of HSMs for this process and the protection of private keys.

Due to the nature of the payments, parties only have a relationship with their immediately preceding and succeeding neighbors. It is for this reason in the traditional approach that neighboring pairs all separately establish a shared symmetric key. For our improved construction this key management requirement remains. As a result we require schemes where a party only requires the help of their immediate neighbors to generate re-encryption keys.

The process for sharing key material between parties must also align closely with the traditional approach due to the requirements mandated by PCI. More specifically, when sharing secret key material they should either be shared in encrypted form or under a strict process of split knowledge. An improvement to this process would be to eliminate the sharing of secret material completely. If we only require parties to share public information to generate re-encryption keys this facilitates a simpler and more secure key ceremony. In Section 5 we discuss how key ceremony processes could be improved in this way using PRE.

3.4.3 Key Rotation

Part of managing the lifecycle of cryptographic keys is to ensure that they can be easily rotated at regular intervals. Key rotation is a vital function as it protects against exposure of data due to overuse of a key and also ensures that we can quickly transition from keys that may have been compromised. Due to the requirement that re-encryption key generation must be performed locally between each neighboring pair, this means that key rotation in the payments setting is relatively straightforward. In both the traditional and our desired settings, we simply locally rotate the key of a party and where appropriate recalculate the re-encryption keys with their neighbors. As there is no global key that needs to be rotated, individual key rotations can happen at each party/pair of parties without requiring coordination with all parties in the system.

4 Cryptographic Definitions and Relations

We introduce the necessary definitions to explain PRE and its security definitions. We discuss issues with the original chosen-plaintext attack (CPA) security definition, the honest re-encryption attack (HRA) strengthening, and its incomparability with chosen-ciphertext attack (CCA) security. As observed in Appendix C, CCA secure schemes are too inefficient for our setting, even though we need a basic level of chosen-ciphertext security, as modeled by our ciphertext-validation attack (CVA) setting. We observe that by focusing on PRE schemes with deterministic re-encryption, HRA and CCA security become comparable, allowing us to introduce hrCCA security, which models HRA security while giving adversaries chosen-ciphertext capability.

4.1 Syntax and Correctness

When clear from context, we do not make explicit the input domains to all algorithms. We assume that if an input is entered into an algorithm, then it is in the correct domain. The length of bitstring x is denoted $|x|$. We interchangeably write $f(a,b,c,\ldots)$ and $f_a(b,c,\ldots)$. Furthermore, for any adversary $\mathbf{A}$, we define

$$\underset{\mathbf{A}}{\Delta}(f_1,f_2,\ldots;g_1,g_2,\ldots) = \left|\mathbf{P}\left[\mathbf{A}^{f_1,f_2,\cdots}=1\right]-\mathbf{P}\left[\mathbf{A}^{g_1,g_2,\cdots}=1\right]\right|, \quad (1)$$

where $\mathbf{A}^{f_1,f_2,\cdots} = 1$ denotes the event where $\mathbf{A}$ outputs 1 when provided oracle access to $(f_1, f_2, \ldots)$. By extension, we let $\Delta_{\mathbf{A}}(G_0\,;G_1)$ denote $\mathbf{A}$'s advantage in distinguishing its interaction with G_0 versus G_1 for two games G_0 and G_1.

Proxy Re-Encryption (PRE) schemes are Public-Key Encryption (PKE) schemes enhanced by a re-encrypt operation. We define PRE schemes in terms of PKE schemes, and therefore recall the definition of a PKE scheme:

Definition 1 (Public-Key Encryption Scheme)**.** *A* Public-Key Encryption scheme *consists of a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KG}, \mathsf{E}, \mathsf{D})$ *with the following interfaces:*

$\mathsf{Setup}(1^\lambda) \to pp$ **: Setup** *on input security parameter λ outputs public parameters pp.*

$\mathsf{KG}(pp) \to (pk, sk)$ **: Key generation** *returns a public and secret key pair (pk, sk).*

$\mathsf{E}(pk, m) \to c$ **: Encryption** *on input public key pk and message m outputs a ciphertext c.*

$\mathsf{D}(sk, c) \to m$ **or** $\perp$ **: Decryption** *on input a secret key sk and a ciphertext c outputs a plaintext message m or error symbol $\perp$.*

PRE enables those holding *re-encryption keys* to transform ciphertext encrypted under a user A's public key to a ciphertext under a different user B's public key. A *bidirectional* PRE scheme [23] permits a re-encryption key $rk_{A\to B}$ to be used to re-encrypt in either direction, i.e. from A to B, or from B to A. In contrast, re-encryption keys generated by *unidirectional* schemes [50] $rk_{A\to B}$ can only be used to re-encrypt in one direction, that is, from A to B, but not from B to A.

Definition 2 (Bidirectional Proxy Re-encryption Scheme)**.** *A* Bidirectional Proxy Re-Encryption scheme *is a public-key encryption scheme with re-key generation* RKG *and re-encryption* RE *functionalities. Re-key generation exposes the following interface:*

$\mathsf{RKG}(sk_i, pk_i, sk_j, pk_j) \to rk_{i\to j}$**:** *on input a source key pair (sk_i, pk_i) and destination key pair (sk_j, pk_j),* RKG *outputs a re-encryption key $rk_{i\to j}$.*

Re-encryption has the following interface:

$\mathsf{RE}(rk_{i\to j}, c) \to c'$ **or** $\perp$**:** *on input a re-encryption key $rk_{i\to j}$ and a ciphertext c,* RE *outputs a transformed ciphertext c' or error symbol, $\perp$.*

Unidirectional schemes are similarly defined, except RKG no longer takes the destination secret key, sk_j, as input. The definition can be found in Appendix D.

Since a PRE scheme is an extension of a PKE scheme, PRE schemes must be correct PKE schemes as well.

Definition 3 (PKE Correctness)**.** *For all possible (pk, sk) generated by* $\mathsf{KG}(pp)$*, for all m, with probability one:*

$$\text{for all } c \leftarrow \mathsf{E}(pk, m) \quad \text{it holds that} \quad m = \mathsf{D}(sk, c)\,. \tag{2}$$

Some PRE schemes restrict how many times a ciphertext can be re-encrypted. Schemes which only permit one re-encryption are *single-hop*, such as the one proposed by Ateniese et al. [15]. *Multi-hop* PRE schemes do not restrict the number of re-encryptions, such as those of Blaze et al. [23] and Canetti and Hohenberger [27].

The correctness definition we give below is in terms of bidirectional rekey generation. The unidirectional counterpart replaces $\mathsf{RKG}(sk_i, pk_i, sk_j, pk_j)$ with $\mathsf{RKG}(sk_i, pk_i, pk_j)$.

Definition 4 (ℓ-hop Bidirectional PRE Correctness)**.** *A bidirectional PRE scheme is ℓ-hop correct if it is correct as a PKE scheme and for all possible (pk_i, sk_i) generated by* $\mathsf{KG}(pp)$ *for $1 \le i \le \ell + 1$, for all $rk_{i\to i+1}$ generated by* $\mathsf{RKG}(sk_i, pk_i, sk_{i+1}, pk_{i+1})$ *for $1 \le i \le \ell$, for all m, with probability one it holds that:*

$$c_1 = \mathsf{E}(pk_1, m) \tag{3}$$

$$c_{i+1} = \mathsf{RE}(rk_{i\to i+1}, c_i) \qquad 1 \le i \le \ell \tag{4}$$

$$m = \mathsf{D}(sk_{\ell+1}, c_{\ell+1}) \qquad . \tag{5}$$

Definition 5 (Multi-hop PRE Correctness)**.** *A PRE scheme is multi-hop correct if it is ℓ-hop correct for all $\ell \ge 1$.*

4.2 PRE Security Definitions

Throughout, we assume that adversaries are computationally-bounded. Furthermore, in-line with most prior work, we consider only static corruptions, meaning the adversary fixes in advance the keys it wishes to expose. An extension of our work could consider security in the presence of adaptive corruptions analogous to Fuchsbauer et al.'s work [43].

In the following subsections we describe the adversary's goal, the oracles it may use, and any restrictions on those oracle queries. Section 4.2.1 reviews the PRE games and settings from the literature. Section 4.2.2 delves into our core observation and contribution, which then allows us to define the new settings we use to model security for PIN translation in Section 4.2.3. Definition 6 formally describes our framework, with oracle definitions given in Figure 4, and Table 1 summarizing what oracles the adversary may access depending on the setting.

4.2.1 Adversarial Goal and Oracle Interaction

We consider standard confidentiality settings, where the adversary's goal is to distinguish the encryption of two equal-length messages m_0 and m_1 of its choosing. Section 5 relates our choice of PRE security definitions to the payment setting.

All games start with a setup phase, where first the adversary decides the total number of users present, then the game generates keys for those users, the adversary divides the users into honest and corrupt subsets, and finally the game exposes the corrupt keys to the adversary. Subsequently, the adversary enters an attack phase where it is given access to PRE functionality depending upon the setting: there are those considered in the past, CPA, CCA, and HRA, and the ones we introduce, hrCVA and hrCCA.

In all settings the adversary has access to a rekey generation oracle and a challenge oracle. We define two Re-Key Generation oracles, one for unidirectional and one for bidirectional schemes. The unidirectional oracle $\mathcal{O}^{\mathsf{uni}}_{\mathsf{RKG}}$ disallows re-key generation if the adversary attempts to generate a key from an honest to a corrupt party. In bidirectional schemes $rk_{i\to j}$ equals $rk_{j\to i}$, hence the bidirectional oracle $\mathcal{O}^{\mathsf{bi}}_{\mathsf{RKG}}$ only permits queries where either both parties are honest or both are corrupt. The challenge oracle $\mathcal{O}_{\mathsf{Chall}}$ may only be queried once; we call its output the *challenge ciphertext*.

In addition to the re-key generation and challenge oracles, the CPA (chosen-plaintext attack) setting allows the adversary to also query re-encryption $\mathcal{O}_{\mathsf{RE}}$. It may not query re-encryption from honest to corrupt users. Cohen [34] points out that CPA secure PRE schemes could be trivially insecure in practice since a guarantee of CPA security does not prevent receivers from learning senders' secret keys. Hence we deem CPA security insufficient and do not consider such schemes.

As a strengthening of CPA, Cohen proposes HRA (honest-re-encryption attack) security [34], where PRE schemes must be secure against adversaries which may re-encrypt from honest to corrupt users, but only ciphertexts which are produced as the result of either an oracle encryption query, $\mathcal{O}_{\mathsf{E}}$, or a re-encryption of an $\mathcal{O}_{\mathsf{E}}$ output. We denote the HRA re-encryption oracle by $\mathcal{O}_{h\mathsf{RE}}$ to distinguish it from the CPA re-encryption oracle $\mathcal{O}_{\mathsf{RE}}$. Cohen proves that HRA implies CPA security.

As we elaborate in Section 5, HRA security does not suffice for our setting since adversaries might have access to a restricted decryption oracle. Hence we look at schemes proven secure in the CCA (chosen-ciphertext attack) setting [27, 56], where adversaries are given access to decryption oracles $\mathcal{O}_{\mathsf{D}}$ for honest users. Yet all known CCA secure schemes are not efficient enough for our application, as we discuss in Appendix C. Instead, we search for a strengthening of HRA which gives adversaries access to a decryption oracle (hrCCA) or a decryption verification oracle (hrCVA).

4.2.2 Understanding CCA Restrictions

Adding a decryption oracle $\mathcal{O}_{\mathsf{D}}$ to the HRA setting without further restrictions allows adversaries to trivially win by decrypting the challenge ciphertext. Similarly, adversaries might try to re-encrypt the challenge ciphertext and then decrypt it. To prevent trivial wins in the CCA setting, Canetti and Hohenberger (CH) [27] introduce the concept of *derived ciphertexts*, where the idea is to "track" the challenge ciphertext as it gets re-encrypted, and forbid the adversary from submitting any ciphertext derived from the challenge ciphertext to $\mathcal{O}_{\mathsf{D}}$.

Defining derived ciphertexts is subtle, since if re-encryption is randomized, there could be many possible ciphertexts that derive from the challenge ciphertext. In particular, if the adversary requests a re-encryption key via $\mathcal{O}_{\mathsf{RKG}}$, then the adversary can re-encrypt with randomness of its choosing thereby making it difficult to track which ciphertexts were derived from the challenge. CH's definition of derived ciphertexts deals with these randomness issues, but as a result CCA and HRA security are incomparable; see Cohen's discussion for details [33, App. B].

We observe that many subtleties disappear when focusing on PRE schemes with *deterministic* re-encryption, where each ciphertext only has one possible re-encryption per re-encryption key. Since our proposed PRE scheme (cf. Section 6.1) has deterministic re-encryption, for our purposes it suffices to focus our formalization on such schemes. We present our definition of derived ciphertexts below, and discuss the differences with CH's definition in Appendix A.

Definition 6 (Derived Ciphertexts)**.** *Let $\mathcal{O}_{\mathsf{RE}}(i,j,c)$ and $\mathcal{O}_{\mathsf{RKG}}(i,j)$ represent the re-encryption and re-key generation oracles exposed to an adversary against a PRE scheme as defined in Figure 4. A ciphertext tuple (i,c) is a* derivative *of another ciphertext tuple (i^*,c^*) if one of the following holds:*

1. *$(i,c) = (i^*,c^*)$ (self or reflexive derivation),*
2. *if (i,c) is a derivative of (i',c'), where (i',c') is a derivative of (i^*,c^*) (transitive derivation),*
3. *if $c = \mathcal{O}_{\mathsf{RE}}(i^*,i,c^*)$ (oracle derivation), or*
4. *if the adversary queries $\mathcal{O}_{\mathsf{RKG}}(i^*,i)$, or equivalently $\mathcal{O}_{\mathsf{RKG}}(i,i^*)$ in the bidirectional case, to get a re-encryption key $rk_{i^*\to i}$ from i^* to i , and $c = \mathsf{RE}(rk_{i^*\to i},c^*)$ (external derivation).*

4.2.3 hrCCA, hrCVA and Formal Definition

With our definition of derived ciphertexts, HRA and CCA are no longer incomparable, and we can extend HRA to a setting which allows the adversary to make queries to the decryption oracle: *hrCCA*. In the remainder of the paper, CH-CCA will refer to the CCA setting using Canetti and Hohenberger's definition of derived ciphertexts, while just CCA refers to the setting we define below.

We furthermore consider the *hrCVA* setting, which more accurately models the requirements of the payments setting; cf. Section 5. The hrCVA setting is a variant of hrCCA, where the adversary has access to a more restricted decryption oracle, $\mathcal{O}_{\mathsf{V}}$, which only checks the validity of a ciphertext.

The fact that CCA implies hrCCA, that hrCCA implies hrCVA, and hrCVA implies HRA security follows trivially from the definitions. Note that all our settings are incomparable with CH-CCA (Canetti Hohenberger CCA, cf. Section 4.2.3), since CH-CCA is incomparable with HRA. Furthermore, CH-CCA is a *replayable* CCA [28] style definition, meaning it tolerates mauling of the ciphertext, whereas our CCA definition does not.

FIGURE 4: PRE GAME ORACLES

```
function $O^{\mathsf{UNI}}_{\mathsf{RKG}}(i, j)$    ▷ Unidirectional Re-key
    if $i \in H, j \in C$ then return $\perp$
    return $\mathsf{PRE.RKG}(sk_i, pk_i, pk_j)$
function $O^{\mathsf{BI}}_{\mathsf{RKG}}(i, j)$    ▷ Bidirectional Re-key
    if $i \in H, j \in C$ then return $\perp$
    if $i \in C, j \in H$ then return $\perp$
    return $\mathsf{PRE.RKG}(sk_i, pk_i, sk_j, pk_j)$
function $O_{\mathsf{Chall}}(i, m_0, m_1)$    ▷ Challenge
    if $i \notin H$ or $|m_0| \neq |m_1|$ then return $\perp$
    return $\mathsf{PRE.E}(pk_i, m_b)$
function $O_{\mathsf{E}}(i, m)$    ▷ Encryption
    return $\mathsf{PRE.E}(pk_i, m)$
function $O_{\mathsf{RE}}(i, j, c)$    ▷ Re-encryption
    if $c \in D^*$ and $j \in C$ then return $\perp$
    return $\mathsf{PRE.RE}(rk_{i \to j}, c)$
function $O_{h\mathsf{RE}}(i, j, c)$    ▷ Honest Re-encryption
    if $c \notin D^* \cup D$, or $c \in D^*$ and $j \in C$ then
        return $\perp$
    return $\mathsf{PRE.RE}(rk_{i \to j}, c)$
function $O_{\mathsf{D}}(j, c)$    ▷ Decryption
    if $c \in D^*$ then return $\perp$
    return $\mathsf{PRE.D}(sk_j, c)$
function $O_{\mathsf{V}}(j, c)$    ▷ Decryption Verification
    if $\mathsf{PRE.D}(sk_j, c) = \perp$ then return false
    return true
```

Definition 7 (PRE-X Advantage)**.** *Consider a proxy re-encryption scheme* $\mathsf{PRE} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{RKG}, \mathsf{RE})$*. Let* $\mathbf{A}$ *be an adversary playing* G_b *denoting an execution of the PRE-X game defined below, with bit* $b \in \{0, 1\}$*. Then* $\mathbf{A}$*'s* PRE-X advantage *for* $X \in \{$*HRA, hrCVA, CCA, hrCCA*$\}$ *is:*

$$\mathbf{Adv}_{\mathit{PRE\text{-}X}}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}(G_0\,;G_1)\ . \tag{6}$$

The game G_b *consists of a* Setup *and* Attack *phase:*

Setup: *Run* $\mathsf{PRE.Setup}$ *and give the generated public parameters to* $\mathbf{A}$*. The adversary* $\mathbf{A}$ *decides n, representing the total number of users, and splits* $\{1, \ldots, n\}$ *into two sets H and C, representing the honest and corrupt users, respectively. Generate n keypairs* $(pk_i, sk_i) \leftarrow \mathsf{PRE.KG}$*, and divide the indices of the keypairs among the sets H and C. For all* $i \in H$ *give* $\mathbf{A}$ pk_i*, and for all* $i \in C$ *give* $\mathbf{A}$ (pk_i, sk_i)*.*

Attack: *For all* $i, j \leq n, i \neq j$*, compute the re-encryption key* $rk_{i \to j}$ *using* $\mathsf{PRE.RKG}$*. Let* D^* *denote the set of ciphertexts which are derived from the output of the challenge oracle* O_{Chall} *(cf. Definition 6);* $D^* = \emptyset$ *before the challenge oracle is queried. Similarly, we let D denote the set of ciphertexts which are derived from any encryption oracle output, with* $D = \emptyset$ *before the encryption oracle is queried. The adversary* $\mathbf{A}$ *is given access to the oracles as defined in Figure 4 and Table 1, where* $\mathbf{A}$ *may query* O_{Chall} *at most once.*

At the end of the attack phase $\mathbf{A}$ *outputs a guess* b'*.*

Table 1: Oracle access for each attack notion. In addition to the oracles specified in the table, the adversary always has access to $O^{\mathsf{uni/bi}}_{\mathsf{RKG}}$ and O_{Chall}.

HRA	hrCVA	hrCCA	CCA
O_{E}	O_{E}	O_{E}	
$O_{h\mathsf{RE}}$	$O_{h\mathsf{RE}}$	$O_{h\mathsf{RE}}$	O_{RE}
	O_{V}	O_{D}	O_{D}

4.3 Hybrid Encryption and PRE

Hybrid-encryption schemes combine public- and symmetric-key encryption: a Key Encapsulation Mechanism (KEM) encrypts a randomly chosen key with public-key encryption, and a Data Encapsulation Mechanism (DEM) encrypts the data under this random key with symmetric-key encryption.

Our construction is an extension of a hybrid-encryption scheme to the PRE setting: when we re-encrypt a ciphertext, we re-encapsulate the KEM output. In preparation, we formally define the components making up *Hybrid-PRE*, namely a DEM and a *key re-encapsulation mechanism* (KREM). In the interest of brevity, and due to their similarity with Definition 3 and 4, the Hybrid-PRE correctness definitions are in Appendix D.

Much as KEMs are similar to PKE schemes, KREMs are similar to PRE schemes with the main difference being that KREMs do not take plaintext as input, and output uniform random keys. As with PRE and PKE, we define KREMs as KEMs with additional functionality.

Definition 8 (Key Encapsulation Mechanism)**.** *A* Key Encapsulation Mechanism (KEM) *consists of a tuple of algorithms* $(\mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps})$ *defined as follows:*

$\mathsf{KG} \to (pk, sk)$: Key generation *returns a public and secret key pair* (pk, sk)

$\mathsf{Encaps}(pk) \to (c, K)$: Encryption *on input the public key pk, outputs a ciphertext c and a key K*

$\mathsf{Decaps}(sk, c) \to K$ or $\perp$: Decryption *on input a secret key sk and a ciphertext c outputs a key K or error symbol* $\perp$*.*

We only define bidirectional KREMs. The definition can be easily modified to capture uni-directional KREMs as well.

Definition 9 (Proxy Key Re-Encapsulation Mechanism)**.** *A* Proxy Key Re-Encapsulation Mechanism (KREM) *is a KEM with the following added functionality:*

$\mathsf{RKG}(sk_i, pk_i, sk_j, pk_j) \rightarrow rk_{i\rightarrow j}$: ReKeyGen *takes as input a source key pair* (sk_i, pk_i) *and a destination key pair* (sk_j, pk_j) *and outputs a re-encapsulation key* $rk_{i\rightarrow j}$.

$\mathsf{ReEncaps}(rk_{i\rightarrow j}, c) \rightarrow c'$ or $\perp$: ReEncapsultion *takes as input a re-encapsulation key* $rk_{i\rightarrow j}$ *and a ciphertext* c *and outputs a transformed ciphertext* c' *or error symbol* $\perp$.

Definition 10 (Data Encapsulation Mechanism)**.** *A* Data Encapsulation Mechanism (DEM) *consists of a tuple of algorithms* $(\mathsf{KG}, \mathsf{E}, \mathsf{D})$ *defined as follows:*

$\mathsf{KG}(pp) \rightarrow K$: Key generation *returns a secret key* K

$\mathsf{E}(K, m) \rightarrow c$: Encryption *on input a key* K *and a message* m*, outputs a ciphertext* c*.*

$\mathsf{D}(K, c) \rightarrow m$ or $\perp$: Decryption *on input a secret key* K *and a ciphertext* c *outputs a message* m *or error symbol* $\perp$.

Definition 11 (DEM-CCA Advantage)**.** *Consider a DEM* $(\mathsf{KG}, \mathsf{E}, \mathsf{D})$*, then the DEM-CCA advantage of an adversary* $\mathbf{A}$ *against it is*

$$\mathbf{Adv}_{DEM}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}(\mathsf{E}_K, \mathsf{D}_K\,;\, \$, \mathsf{D}_K)\ , \tag{7}$$

where $K \xleftarrow{\$} \mathsf{KG}$*,* $\$(x)$ *returns independent, uniform random strings of length* $|E_K(x)|$*, and* $\mathbf{A}$ *may query its left oracle only once and may not use the output of its left oracle as input to its right oracle.*

KREM security can be defined in the same settings as PRE security, but we only need to focus on hrCCA KREM security. The *KREM-hrCCA* game is the same as our PRE hrCCA game, except encryption, re-encryption, and decryption are replaced by encapsulation, re-encapsulation, and decapsulation. We also use the same concept of derived ciphertexts to prevent trivial wins. The definition of KREM-hrCCA advantage can be found in Appendix D under Definition 19, with oracle definitions found in Figure 8.

The following theorem follows from a straightforward hybrid argument, similar to how Giacon, Kiltz, and Poettering [47] prove the same result for traditional hybrid-encryption in the multi-user setting.

Theorem 1. *Combining an hrCCA KREM with a CCA DEM results in an hrCCA PRE scheme, i.e. we can construct efficient adversaries* $\mathbf{B}$ *and* $\mathbf{C}$ *such that*

$$\mathbf{Adv}_{PRE}(\mathbf{A}) \leq 2 \cdot \mathbf{Adv}_{KREM}(\mathbf{B}) + \mathbf{Adv}_{DEM}(\mathbf{C})\,. \tag{8}$$

5 PIN Translation using PRE

Recall that our objective is to transmit a PIN from the PoS terminal to the consumer bank, while ensuring confidentiality of the PIN from the intermediate parties. Using a PRE scheme which satisfies our PRE-CVA notion gives this guarantee and mitigates the threats outlined in Section 3.1. The reason we only consider CVA attacks is because the final PIN verification only outputs a true/false whether the ciphertext is valid and the PIN matched or not. Thus we do need to be concerned with adversaries having access to a full decryption oracle. In the rest of this section we describe how the various PRE algorithms would be used by each participant in the payment ecosystem.

5.1 General Construction

Since banks and payment gateways join the payments ecosystem dynamically (i.e. the network is not static), and since PoS terminals are deployed on demand, we decompose our system based on two processes: enrollment of new parties and processing online transactions.

We assume an initial setup has been performed, where a PRE scheme $\mathsf{PRE} = (\mathsf{Setup}, \mathsf{KG}, \mathsf{E}, \mathsf{D}, \mathsf{RE})$ is chosen, and $\mathsf{PRE.Setup}$ invoked to generate public parameters for use in all future PRE operations. These parameters must be shared with all parties in the ecosystem, and therefore, it is ideally generated by the network processors and forms part of some global standard.

5.1.1 Enrollment

Network processors (e.g., Visa) continuously add merchant and consumer banks to their network, and are responsible for managing keys to route transactions between any pair of banks; similarly, merchant banks enroll new payment gateways dynamically, and are responsible for key management for all PoS devices. To that end, we introduce an *HSM-backed key manager* at each network processor, merchant bank, and consumer bank following the requirements presented in Section 3.4 – note that the key managers are only used during key ceremonies for generating key material, and not during payment transactions (thus reducing our reliance on HSMs).

Here we elaborate on how a bidirectional PRE scheme can be applied to perform PIN translation. First we focus on the initial key management steps that must be performed.

To enroll a bank, the network processor generates a keypair using $\mathsf{PRE.KG}$ and shares the corresponding private key with that bank's trusted key manager (using the *split knowledge* approach in Section 3.4). Using its knowledge of the merchant bank's secret key sk_{mb} and consumer bank's key sk_{cb}, the network processor's key manager uses $\mathsf{PRE.RKG}$ to generate a re-encryption key $rk_{mb\rightarrow cb}$ for its use while processing transactions. The merchant bank enrolls each payment gateway and its set of PoS devices by generating a gateway-specific keypair $(pk_{gw}, sk_{gw}) \leftarrow \mathsf{PRE.KG}$, and sharing pk_{gw} with the PoS partner, and sharing the re-encryption key $rk_{gw\rightarrow mb}$ (derived from the gateway-specific sk_{gw} and the merchant bank's own secret key sk_{mb}) with the gateway – note that sk_{mb} is not shared with any party.

5.1.2 PIN-based Transactions

Following the above key management steps, public keys and re-encryption keys will have been distributed appropriately to all parties and now during the transaction only the consumer bank needs an HSM (to perform a decryption and a PIN verification operation). More specifically each transaction proceeds as follows:

1. Using the pk_{gw} provided by the merchant bank/PoS partner, the PoS device invokes $\mathsf{PRE.E}$ on the user-supplied PIN and sends the encrypted pin block to the gateway.
2. The payment gateway uses its re-encryption key $rk_{gw \to mb}$ to invoke $\mathsf{PRE.RE}$, and send the re-encrypted PIN to the merchant bank.
3. The merchant bank forwards the incoming ciphertext to the network processor verbatim.
4. The network processor determines the consumer bank that must authorize the transaction. It then looks up the corresponding re-encryption key $rk_{mb \to cb}$, invokes $\mathsf{PRE.RE}$, and sends on the re-encrypted PIN block.
5. The consumer bank's HSM uses the private key sk_{cb} to invoke $\mathsf{PRE.D}$ (producing the cleartext PIN within HSM), and authenticates the PIN based on some additional secret data needed for PIN verification.[3]

5.1.3 Bi-directional versus Uni-directional

Above we have focused on how to deploy a bidirectional PRE scheme for PIN translation and its corresponding key management. As a bidirectional scheme requires the secret keys of both parties to compute a re-encryption key, it is necessary to follow a strictly controlled process during initial setup. During enrollment/setup we need to securely share secret keys between parties (as required) just as in the traditional approach (cf. Section 3.4.1). However, after the key ceremony any involved HSMs are no longer required and hence will remain offline (unlike the traditional approach where HSMs are consistently online increasing their chance of exposure).

A further concern when using a bidirectional scheme is that leakage of a single secret key would permit an attacker with knowledge of the re-encryption keys to infer all other secret keys. Due to the fact the HSMs are only online during a transaction and also due to fact they are operated in a strictly controlled and audit manner, the risk of such an exposure is low. A simple mitigation which eliminates the risk is to delete secret key material after re-encryption keys are generated. This would however come at the expense of complicating the enrollment of new intermediaries.

The use of unidirectional PRE scheme, would both simplify the key-ceremony process and also mitigate the risk of key inference. A unidirectional scheme only needs the source secret key and destination public key to generate a re-encryption key. Sharing of the destination public key with the source can be performed in a more relaxed manner since there is little concern related to its exposure.

When a new participant enrolls, their trusted key manager (HSM) generates a key pair using $\mathsf{PRE.KG}$. Now to generated a re-encryption key from Party A to Party B, Party B shared its public key with Party A. Then Party A used its secret key together with B's public to generate a re-encryption key from A to B. In this setting, as only public keys are shared between participants we do not need to use KEKs or split knowledge during key transfer, making key ceremonies much simpler. Due to the variation in re-encryption key setup, transactions based on unidirectional PRE proceed slightly differently. In the interests of space we detail a uni-directional PRE-based transaction in Appendix B.

Despite the attractive key management characteristics of unidirectional schemes all known constructions lack the level of efficiency we require for our setting. As a result in the rest of this paper we focus on how to construct a solution based on bidirectional PRE.

6 Hybrid PRE Construction

Appendix C surveys known schemes to determine their suitability. We find that no published PRE schemes satisfy our criteria, and introduce our construction.

6.1 Our Construction

We present our Hybrid PRE scheme and analyze its security. It is formed of a KREM, used to share a randomly chosen key, and a DEM, which encrypts the PIN under a random key protected by the KREM. Re-encryption changes the KREM output but leaves the DEM output unchanged.

As with other PRE schemes and hybrid PKE schemes [36], the basis of our scheme is ElGamal encryption [44]. Our scheme, given in Figure 5, works over a group of prime order q with generator g and uses a symmetric encryption scheme $\mathsf{SE} = (\mathsf{SE.E}, \mathsf{SE.D})$. Key Generation selects a secret key sk at random from $\mathbb{Z}_q^*$ and calculates the public key $pk = g^{sk}$. Our construction is bidirectional so re-encryption key generation $\mathsf{PRE.RKG}$ requires both secret keys, sk_i and sk_j. The re-encryption key from party i to party j is sk_i/sk_j.

Encryption $\mathsf{PRE.E}$ first performs the KREM, where a random exponent r is chosen and $c_1 = g^r$ calculated. The symmetric key K is the hash of $pk^r = g^{sk \cdot r}$; we effectively have an ephemeral-static Diffie-Hellman key. Next, the DEM runs $\mathsf{SE.E}$ on the input message m with the key K, to obtain c_2. The output of $\mathsf{PRE.E}$ is (c_1, c_2).

Decryption follows the reverse procedure. First, c_1 is decapsulated to obtain the symmetric key K, by taking the hash of $c_1^{sk} = g^{r \cdot sk}$. Then using K we call $\mathsf{SE.D}$ on c_2 to retrieve m. As mentioned previously, the re-encryption function $\mathsf{PRE.RE}$

[3] PIN verification is generally performed using the IBM3624 method. This is based on the cleartext PIN, a secret PIN verification key and a PIN offset. As we are focused on PIN privacy we omit full details on how PIN verification is performed.

Figure 5: Hybrid-PRE Construction

```
function PRE.Setup($1^\lambda$)                              ▷ Setup
    Choose $\mathbb{G}$ of prime order $q$ with generator $g$
    return $pp = (g, q)$
function PRE.KG($pp$)                                        ▷ Key Generation
    $sk \xleftarrow{\$} \mathbb{Z}_q^*$; $pk \leftarrow g^{sk}$
    return $(pk, sk)$
function PRE.RKG($(pk_i, sk_i), (pk_j, sk_j)$)
                                                             ▷ ReKey Generation
    $rk_{i\to j} \leftarrow sk_i / sk_j$
    return $rk_{i\to j}$
function PRE.E($pk, m$)                                      ▷ Encryption
    $r \xleftarrow{\$} \mathbb{Z}_q^*$; $c_1 \leftarrow g^r$
    $K = F(pk^r)$; $c_2 \leftarrow$ SE.E($K, m$)
    return $(c_1, c_2)$
function PRE.D($sk, c$)                                      ▷ Decryption
    Parse $c$ as $(c_1, c_2)$
    $K = F(c_1^{sk})$; $m \leftarrow$ SE.D($K, c$)
    return $m$
function PRE.RE($rk_{i\to j}, c$)                            ▷ ReEncryption
    Parse $c$ as $(c_1, c_2)$
    $c_1' \leftarrow c_1^{rk_{i\to j}}$
    return $(c_1', c_2)$
```

Figure 6: KREM Construction

```
function PRE.Setup($1^\lambda$)                              ▷ Setup
    Choose $\mathbb{G}$ of prime order $q$ with generator $g$
    return $pp = (g, q)$
function PRE.KG($pp$)                                        ▷ Key Generation
    $sk \xleftarrow{\$} \mathbb{Z}_q^*$; $pk \leftarrow g^{sk}$
    return $(pk, sk)$
function PRE.RKG($(pk_i, sk_i), (pk_j, sk_j)$)
                                                             ▷ ReKey Generation
    $rk_{i\to j} \leftarrow sk_i / sk_j$
    return $rk_{i\to j}$
function PRE.Encaps($pk$)                                    ▷ Encapsulation
    $r \xleftarrow{\$} \mathbb{Z}_q^*$; $c_1 \leftarrow g^r$
    $K = F(pk^r)$;
    return $(c_1, K)$
function PRE.Decaps($sk, c_1$)                               ▷ Decapsulation
    $K = F(c_1^{sk})$;
    return $K$
function PRE.ReEncaps($rk_{i\to j}, c_1$)                    ▷ ReEncaps
    $c_1' \leftarrow c_1^{rk_{i\to j}}$
    return $c_1'$
```

only operates on the KREM output c_1. Here re-encryption from party i to party j is performed by raising it to the re-encryption key $c_1^{rk_{i\to j}}$.

Figure 5 uses a generic symmetric encryption scheme SE as the DEM. The actual DEM we implement is based on standard PIN Block encryption (cf. Figure 2 and 3), which encodes the PIN based on ISO 9564-1 and then encrypts this PIN Block using 3DES using the random key protected by the KREM. This means that our DEM ensures best possible backwards compatibility.

6.2 Security Analysis

We analyze the security of our construction by analyzing the security of the Hybrid-PRE construction presented in Section 6.1 and also the DEM-CCA security of the EPB construction from Figure 3.

We rely on the gap-Strong-Diffie-Hellman (gap-SDDH) assumption in the random oracle model. This problem is the same as the ODH-RO problem [11], with the addition of $g^{1/v}$, or alternatively the SDDH assumption [64] with an oracle which returns random oracle output evaluated on X^v.

Definition 12 (gap-SDDH-RO Advantage)**.** *An adversary* **A***'s advantage in playing the gap-SDDH-RO game is*

$$\underset{\mathbf{A}}{\Delta}\left(F, F_v, g^u, g^v, g^{1/v}, F(g^{uv}); F, F_v, g^u, g^v, g^{1/v}, W\right), \qquad (9)$$

where F is a random oracle, u, v, W are chosen uniformly at random, $F_v(X) := F(X^v)$, and **A** *may not call F_v on g^u.*

The KREM underlying our Hybrid PRE construction in Figure 5 is given in Figure 6, cf. Appendix D. It performs the same operations as the main PRE construction, except it does not take a plaintext as input, nor does it call the DEM. During encapsulation it outputs the key K alongside c_1.

Theorem 2. *Let* **A** *be a KREM hrCCA adversary against the KREM presented in Figure 6, then we construct a gap-SDDH-RO adversary* **B** *such that*

$$\mathbf{Adv}_{KREM}(\mathbf{A}) \le \mathbf{Adv}_{gap\text{-}SDDH\text{-}RO}(\mathbf{B}), \qquad (10)$$

where if **A** *takes time t, then* **B** *takes at most time $O(n \cdot t)$ with n the number of users.*

The proof of Theorem 2 is given in Appendix E.

Next we show that the standard PIN block encryption scheme (cf. Figure 2 and 3) is a secure DEM. This result nearly follows from the Encode-then-Encipher paradigm of Bellare and Rogaway [21], with two main differences: 1) we must now consider associated data,[4] and 2) due to our use of hybrid encryption, we need only consider one-time use of the key and cipher. The effect of this last point is that we no

[4] The encoding scheme for PIN blocks uses the PAN as additional associated data. For simplicity, we have omitted this from our earlier definitions but our results would continue to hold with its addition.

longer need to consider the collision resistance of the encoding scheme since there will only ever be one encrypt call for each DEM key.

Theorem 3. *EPB's* SE *(as defined in Figure 3) is DEM-CCA, i.e., given a DEM-CCA adversary* **A***, we can construct efficient adversaries* **B** *and* **C** *(each permitted only one query to their respective oracle) such that*

$$\mathbf{Adv}_{DEM}(\mathbf{A}) \leq \mathbf{Adv}_{\mathsf{prf}}(\mathbf{B}) + 2 \cdot \mathbf{Adv}_{\mathsf{sprp}}(\mathbf{C}) + 4 \cdot \delta.$$

where $\delta \leq 2^{-48}$ *for ISO-0 and* $\delta \leq 2^{-45}$ *for ISO-3 encodings, and* $\mathbf{Adv}_{\mathsf{prf}}$ *and* $\mathbf{Adv}_{\mathsf{sprp}}$ *are the PRF and SPRP advantages against E as defined in Definition 20 and Definition 21.*

This is proven by extension of results by Bellare and Rogaway [21]. Full details are given in Appendix F.

Finally we apply Theorem 1 and due to hrCCA implying hrCVA, our construction is secure for our payment setting.

7 Implementation and Evaluation

We study the time overhead (latency and throughput) and the space overhead of our PRE-based scheme, and compare with the HSM-based approaches described below.

Deployment. We consider the potential for phased rollout of the PRE-based scheme, where only a subset of the parties adopt the PRE scheme while the remaining parties use the HSM-based approach. Each phase is then a step towards an all PRE-based deployment. We study the following deployments:

1. **All HSM**: This is the current deployment where all parties use the HSM-based approach.
2. **PoS-MB**: This is a phased deployment, where the PoS performs PRE encryption, gateway performs re-encryption, and the merchant bank computes the PRE decryption and 3DES encryption within an HSM — the network processor and consumer bank perform HSM-based encryption and decryption operations.
3. **PoS-Proc**: Similar to PoS-MB, except now the network processor also migrates to the PRE-based approach. Specifically, the PoS perform PRE encryption, gateway performs re-encryption, the merchant bank forwards, and the network processor computes the PRE decryption and encryption within an HSM — the consumer bank decrypts within an HSM.
4. **All PRE**: All parties migrate to the PRE-based approach, where the PoS performs PRE encryption, gateway and network processors perform re-encryption, and the consumer bank performs PRE decryption (merchant bank simply forwards).

Setup. We simulate how PIN translation operates in a payments network by implementing applications for each of the participants. We implement a mobile PoS application on the Android platform which receives the user's PIN and initiates the PIN-based transaction; we deployed our mobile PoS application on Honor 7X equipped with Android 8.0. We implement web applications for the payment gateway service, merchant bank, network processor, and consumer bank, all using dropwizard [38]. The web applications use LevelDB [46] to efficiently manage re-encryption keys in memory.

In reality, the parties in the payments ecosystem are geographically distributed and the wide-area network links dominate the latency of the cryptographic operations. To focus on comparing the latency of the schemes we evaluate the system, containing the PoS and web applications, on a single machine. We use a server with 12 physical cores (Intel Xeon E5-2650 V4, 2.2GHz), 128GB memory, and 2.8 TBs of SSD storage.

We implement our PIN translation scheme based on the hybrid-PRE construction shown in Figure 5 with the DEM based on standard PIN Block encryption (Section 2.3). The ISO 8583 [49] messaging standard has fields reserved for private use [66], which we can use to store the KREM component of the PRE ciphertext (i.e., g^r in Figure 5) of size 32 bytes — the unmodified DEM component of the ciphertext uses the same field as used by the current deployment. We use NIST P-256 curve (secp256r1) [30] with the parameters recommended by the standard. We implement the scheme in Java, using the Bouncy Castle API [7].

For performance measurements of deployments with HSMs at one or more parties (namely, All HSM, Pos-MB, and Pos-Proc), we provision a Thales payShield 9000 HSM in a restricted network zone, and have the web applications invoke operations over the network. This emulates typical deployment of HSMs in payments.

Party	All HSM	All PRE	PoS-MB	PoS-Proc
PoS	98	348	348	348
Gateway	920	161	161	161
Mer. Bank	920	—	934	—
Net Proc.	920	161	920	934
Cons. Bank	900	934	900	900

(a) Latency measurements (in microseconds)

Party	All HSM	All PRE	PoS-MB	PoS-Proc
Gateway	1086	6240	6240	6240
Mer. Bank	1086	—	1025	—
Net Proc.	1086	6240	1086	1025
Cons. Bank	1110	1025	1110	1110

(b) Throughput (in txs / sec)

Figure 7: Performance Results

Latency. In Figure 7a, we compare the latency of all deployment approaches. For the All-PRE deployment, the latency measurement includes the PRE.E operation at the PoS terminal, PRE.RE at the gateway and network processor, and

PRE.D at the consumer bank (in addition to the PIN verification) — the merchant bank simply forwards the incoming message and has negligible latency. In HSM-based deployments, we report both the time spent locally within the HSM — to perform a 3DES decrypt followed by 3DES encrypt — and the remote invocation over the network from the application. The latency characteristics in phased deployments depend on whether the party uses HSM or PRE, i.e., the measurement is similar to either All-HSM or All-PRE, with the caveat of the merchant bank in PoS-MB and the network processor in Pos-Proc — in these two cases, the merchant bank and the network processor performs PRE decryption and 3DES encryption within an HSM, thus incurring longer latency.

Apart from the encryption at PoS terminal, we find that the PRE-based approach provides much lower latency, mainly because the operation can be executed locally by the application — the roundtrip for HSM operations incurred approximately 750 μs in our experiments. While PRE-based PIN encryption is more expensive for the PoS terminal (2 group exponentiations), in comparison with one 3DES encryption, this latency difference is more than offset by the latency savings at other parties processing that transaction. Recall that verification at the consumer bank requires an HSM in both the traditional and the PRE-based schemes; therefore, the consumer bank's latency for all deployments is roughly similar.

Throughput. We measure throughput by launching 100 threads concurrently to generate PIN-based transactions, and report the results in Figure 7b. Our experiment shows that the PRE scheme on a commodity server can handle the average throughput of network processors such as Visa (roughly 5000 txs / sec [6]), and achieves roughly 5× the throughput of a single HSM that is dedicated to PIN translations. The caveat is that even the PRE-based approach requires an HSM at the consumer bank, and therefore shows similar throughput as the HSM-based approach — while it may appear from this that the throughput is bounded by the consumer bank in all deployments, we recall that a typical network processor services thousands of consumer banks and is therefore the bottleneck for throughput (making our 5x improvement worthwhile).

In both HSM-based and PRE-based systems, throughput scales linearly with the amount of hardware parallelism. In practice, HSMs are deployed in a restricted environment, and the throughput is also bound by network latency; on the other hand, our PRE-based approach scales better as operations are invoked locally by the application. It is possible to achieve desired throughput by deploying multiple HSMs and process transactions by invoking HSMs concurrently. However, it would be expensive for gateways and network processors to procure and manage HSMs. For example, the hourly price of Amazon CloudHSM [16] is $1.81, 9 times the cost of a general purpose large Amazon EC2 instance [17] ($0.1984). Furthermore, private cloud deployments would be more expensive due to additional operational overheads and the cost of specialized payment HSMs.

Storage. The network processor (e.g. Visa) endures the largest, yet practically reasonable storage requirement — in the worst case, with m merchant banks and n consumer banks, the network processor must store mn re-encryption keys. For example, there are approximately 15900 financial institutions currently active with Visa [6], in which case our scheme requires approximately 7.5GB of (unprotected) storage as opposed to approximately 0.48MB of HSM protected memory. The PoS device requires 32 bytes to store the merchant bank's public key. The payment gateway stores a 32 byte re-encryption key for each merchant bank, and requires roughly 500 KB in the current state of the payments ecosystem. The merchant bank does not need any storage for key material. The consumer bank stores a 32 byte decryption key within its HSM.

8 Related Work

Berkman and Ostrovsky [22] show how anyone with HSM access at either the consumer bank or an intermediary can obtain customer PINs. Mannan and van Oorschot [58, 59] propose a fix called 'salted-PIN' but it requires significant changes to the payments infrastructure. Steel [70] and Centenaro et al. [29] formally analyze HSM APIs for PIN verification and PIN translation to check for vulnerabilities.

Jayasinghe et al. [51] propose several solutions to online PIN verification which avoid the need for PIN translation and reduce the trust needed in intermediaries. However, much like the alternatives we explore in Section 3.3, their proposals would require significant changes to the payment card and EMV standard.

Myers and Shull [60] previously studied constructions for hybrid PRE. Their work focuses on use cases related to key rotation for cloud outsourced data storage and access control. In this context *Key-scraping* attacks are a major concern. Here, an adversary initially has access to the data and at this time takes note of the random key encapsulated in the KEM. Following a key rotation and re-encryption, it is assumed that the adversary should no longer have access to the plaintext. However, in the standard approach, re-encryption is only performed on the KEM, leaving the DEM the same even after re-encryption. This means that an adversary, who previously recorded the random key held in the KEM, will always be able to decrypt the "re-encrypted" data.

In our setting key-scraping attacks are not a concern primarily due to the ephemeral nature of the transactions and the party performing decryption. As mandated by PCI [63], a fresh random DEM key must be used for each encrypted PIN block and after verification the encrypted PIN block is discarded. The only party that decapsulates to obtain the (ephemeral) DEM key is the consumer bank performing verification, shortly after which the PIN block is destroyed. For an adversary to obtain a DEM key at an intermediary earlier in the transaction would constitute a major attack since that implies a compromise of their long-term secret key.

9 Conclusion

In this paper, we study how proxy re-encryption can be used to reduce the reliance on HSMs for PIN translation in payment transactions. We introduce a new notion for PRE, namely Ciphertext-Verification Attack which is weaker than CCA security but more closely matches the attack model of secure payments. Our final construction is based on a bidirectional PRE scheme which can be efficiently realized and makes use of a hybrid approach to ensure the encryption of the PIN itself remains backwards compatible with existing PIN blocks.

An extension of our work would perform a complete analysis of authentication by studying PIN verification in conjunction with EMV cryptogram verification [40]. It also remains an open problem to construct an efficient multi-hop unidirectional PRE-CVA scheme. Known constructions are based on lattices [31, 52, 67] or generically from Fully-Homomorphic Encryption [45], but they incur significant overheads.

References

[1] A Guide to EMV Chip Technology. Date Accessed: Oct-2020, https://www.emvco.com/wp-content/uploads/2017/05/A_Guide_to_EMV_Chip_Technology_v2.0_20141120122132753.pdf.

[2] Debit card interchange fees and routing. Date Posted: Dec-2010, https://www.govinfo.gov/content/pkg/FR-2010-12-28/pdf/2010-32061.pdf.

[3] EMV at the pump. Date Posted: May-2019, https://usa.visa.com/visa-everywhere/security/emv-at-the-pump.html.

[4] EMV in the U.S.: Putting It into Perspective for Merchants and Financial Institutions. Date Accessed: October-2019, https://www.firstdata.com/downloads/thought-leadership/EMV_US.pdf.

[5] EMV Newsletters 2017. Date Posted: Oct-2017, https://usa.visa.com/dam/VCOM/regional/na/us/run-your-business/documents/emv-newsletter-oct2017.pdf.

[6] Fact Sheet - Visa. Date posted: June-2018, https://usa.visa.com/dam/VCOM/download/corporate/media/visanet-technology/aboutvisafactsheet.pdf.

[7] The Bouncy Castle Cryptographic APIs. https://github.com/bcgit/bc-java.

[8] Visa chip card update. Date Posted: June-2017, https://usa.visa.com/visa-everywhere/blog/bdp/2019/05/28/chip-technology-helps-1559068467332.html.

[9] Visa Minimum U.S. Online Only Terminal Configuration. Date Posted: Dec-2017, https://usa.visa.com/dam/VCOM/regional/na/us/run-your-business/documents/visa-minimum-us-online-only-terminal-configuration.pdf.

[10] ISO 9564-1:2017, Financial services – Personal Identification Number (PIN) management and security – Part 1: Basic principles and requirements for PINs in card-based systems, November 2017.

[11] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle diffie-hellman assumptions and an analysis of DHIES. In David Naccache, editor, *Topics in Cryptology - CT-RSA 2001, The Cryptographer's Track at RSA Conference 2001, San Francisco, CA, USA, April 8-12, 2001, Proceedings*, volume 2020 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2001.

[12] Michel Abdalla and Ricardo Dahab, editors. *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I*, volume 10769 of *Lecture Notes in Computer Science*. Springer, 2018.

[13] American National Standard for Financial Services. ANSI X9.24-1-2017, Retail Financial Services Symmetric Key Management Part 1: Using Symmetric Techniques, June 2017.

[14] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-private proxy re-encryption. In Marc Fischlin, editor, *Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings*, volume 5473 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2009.

[15] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.

[16] Amazon AWS. AWS CloudHSM Pricing, 6 Oct 2020. https://aws.amazon.com/cloudhsm/pricing/.

[17] Amazon AWS. AWS On-Demand Pricing, 6 Oct 2020. https://aws.amazon.com/ec2/pricing/on-demand/.

[18] Sebastian Banescu, Martín Ochoa, Nils Kunze, and Alexander Pretschner. Idea: Benchmarking indistinguishability obfuscation - A candidate implementation. In Frank Piessens, Juan Caballero, and Nataliia Bielova, editors, *Engineering Secure Software and Systems - 7th International Symposium, ESSoS 2015, Milan, Italy,*

March 4-6, 2015. Proceedings, volume 8978 of *Lecture Notes in Computer Science*, pages 149–156. Springer, 2015.

[19] Elaine Barker. Recommendation for Key Management: Part 1 — General, May 2020. https://doi.org/10.6028/NIST.SP.800-57pt1r5.

[20] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[21] Mihir Bellare and Phillip Rogaway. Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 317–330. Springer, 2000.

[22] Omer Berkman and Odelia Moshe Ostrovsky. The unbearable lightness of PIN cracking. In Sven Dietrich and Rachna Dhamija, editors, *Financial Cryptography and Data Security, 11th International Conference, FC 2007, and 1st International Workshop on Usable Security, USEC 2007, Scarborough, Trinidad and Tobago, February 12-16, 2007. Revised Selected Papers*, volume 4886 of *Lecture Notes in Computer Science*, pages 224–238. Springer, 2007.

[23] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 1998.

[24] Alexander Block. BLS: Is it really that slow?, Jul 2018. Date Accessed: Aug 29, 2020. https://blog.dash.org/bls-is-it-really-that-slow-4ca8c1fcd38e.

[25] Dan Boneh, Sergey Gorbunov, Riad S. Wahby, Hoeteck Wee, and Zhenfei Zhang. draft-irtf-cfrg-bls-signature-02. Internet-Draft draft-irtf-cfrg-bls-signature-02, Internet Engineering Task Force, March 2020. Work in Progress.

[26] Cristian Borcea, Arnab Deb Gupta, Yuriy Polyakov, Kurt Rohloff, and Gerard W. Ryan. PICADOR: end-to-end encrypted publish-subscribe information distribution with proxy re-encryption. *Future Generation Comp. Syst.*, 71:177–191, 2017.

[27] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 185–194. ACM, 2007.

[28] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2003.

[29] Matteo Centenaro, Riccardo Focardi, Flaminia L. Luccio, and Graham Steel. Type-based analysis of PIN processing APIs. In Michael Backes and Peng Ning, editors, *Computer Security - ESORICS 2009, 14th European Symposium on Research in Computer Security, Saint-Malo, France, September 21-23, 2009. Proceedings*, volume 5789 of *Lecture Notes in Computer Science*, pages 53–68. Springer, 2009.

[30] Certicom Research. Standards for Efficient Cryptography, SEC 2: Recommended Elliptic Curve Domain Parameters. http://www.secg.org/sec2-v2.pdf.

[31] Nishanth Chandran, Melissa Chase, Feng-Hao Liu, Ryo Nishimaki, and Keita Xagawa. Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In Krawczyk [53], pages 95–112.

[32] Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient unidirectional proxy re-encryption. In Daniel J. Bernstein and Tanja Lange, editors, *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010. Proceedings*, volume 6055 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2010.

[33] Aloni Cohen. What about bob? the inadequacy of CPA security for proxy reencryption. *IACR Cryptology ePrint Archive*, 2017:785, 2017.

[34] Aloni Cohen. What about bob? the inadequacy of CPA security for proxy reencryption. In Lin and Sako [57], pages 287–316.

[35] Craig Costello. Pairings for beginners. Date Accessed: Aug 29, 2020. http://www.craigcostello.com.au/pairings/PairingsForBeginners.pdf.

[36] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

[37] Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen. Chosen-ciphertext secure proxy re-encryption without pairings. In Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong, editors, *Cryptology and Network Security, 7th International Conference, CANS 2008, Hong-Kong, China, December 2-4, 2008. Proceedings*, volume 5339 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.

[38] Dropwizard. A damn simple library for building production-ready RESTful web services. https://github.com/dropwizard/dropwizard.

[39] Priyanka Dutta, Willy Susilo, Dung Hoang Duong, Joonsang Baek, and Partha Sarathi Roy. Lattice-based unidirectional ibpre secure in standard model, 2020.

[40] EMVco. EMV Integrated Circuit Card Specifications for Payment Systems, Book 2 - Security and Key Management Version 4.3, August 2011.

[41] Xiong Fan and Feng-Hao Liu. Proxy re-encryption and re-signatures from lattices. *IACR Cryptology ePrint Archive*, 2017:456, 2017.

[42] Xiong Fan and Feng-Hao Liu. Proxy re-encryption and re-signatures from lattices. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, volume 11464 of *Lecture Notes in Computer Science*, pages 363–382. Springer, 2019.

[43] Georg Fuchsbauer, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Adaptively secure proxy re-encryption. In Lin and Sako [57], pages 317–346.

[44] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18. Springer, 1984.

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

[46] Sanjay Ghemawat and Jeff Dean. Leveldb, 2011.

[47] Federico Giacon, Eike Kiltz, and Bertram Poettering. Hybrid encryption in a multi-user setting, revisited. In Abdalla and Dahab [12], pages 159–189.

[48] Goichiro Hanaoka, Yutaka Kawai, Noboru Kunihiro, Takahiro Matsuda, Jian Weng, Rui Zhang, and Yunlei Zhao. Generic construction of chosen ciphertext secure proxy re-encryption. In Orr Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, volume 7178 of *Lecture Notes in Computer Science*, pages 349–364. Springer, 2012.

[49] ISO-8583. Financial transaction card originated messages — Interchange message specifications. https://www.iso.org/obp/ui/#iso:std:iso:8583:-1:en.

[50] Anca-Andreea Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*. The Internet Society, 2003.

[51] Danushka Jayasinghe, Raja Naeem Akram, Konstantinos Markantonakis, Konstantinos Rantos, and Keith Mayes. Enhancing EMV online PIN verification. In *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*, pages 808–817. IEEE, 2015.

[52] Elena Kirshanova. Proxy re-encryption from lattices. In Krawczyk [53], pages 77–94.

[53] Hugo Krawczyk, editor. *Public-Key Cryptography - PKC 2014 - 17th International Conference on Practice and Theory in Public-Key Cryptography, Buenos Aires, Argentina, March 26-28, 2014. Proceedings*, volume 8383 of *Lecture Notes in Computer Science*. Springer, 2014.

[54] Junzuo Lai, Zhengan Huang, Man Ho Au, and Xianping Mao. Constant-size cca-secure multi-hop unidirectional proxy re-encryption from indistinguishability obfuscation. In Willy Susilo and Guomin Yang, editors, *Information Security and Privacy - 23rd Australasian Conference, ACISP 2018, Wollongong, NSW, Australia, July 11-13, 2018, Proceedings*, volume 10946 of *Lecture Notes in Computer Science*, pages 805–812. Springer, 2018.

[55] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In Ronald Cramer, editor, *Public Key Cryptography - PKC 2008,*

11th International Workshop on Practice and Theory in Public-Key Cryptography, Barcelona, Spain, March 9-12, 2008. Proceedings, volume 4939 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2008.

[56] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Trans. Information Theory*, 57(3):1786–1802, 2011.

[57] Dongdai Lin and Kazue Sako, editors. *Public-Key Cryptography - PKC 2019 - 22nd IACR International Conference on Practice and Theory of Public-Key Cryptography, Beijing, China, April 14-17, 2019, Proceedings, Part II*, volume 11443 of *Lecture Notes in Computer Science*. Springer, 2019.

[58] Mohammad Mannan and Paul C. van Oorschot. Weighing down "the unbearable lightness of PIN cracking". In Gene Tsudik, editor, *Financial Cryptography and Data Security, 12th International Conference, FC 2008, Cozumel, Mexico, January 28-31, 2008, Revised Selected Papers*, volume 5143 of *Lecture Notes in Computer Science*, pages 176–181. Springer, 2008.

[59] Mohammad Mannan and Paul C. van Oorschot. Reducing threats from flawed security apis: The banking PIN case. *Computers & Security*, 28(6):410–420, 2009.

[60] Steven Myers and Adam Shull. Efficient hybrid proxy re-encryption for practical revocation and key rotation. *IACR Cryptology ePrint Archive*, 2017:833, 2017.

[61] David Nuñez, Isaac Agudo, and Javier López. Ntrureencrypt: An efficient proxy re-encryption scheme based on NTRU. In Feng Bao, Steven Miller, Jianying Zhou, and Gail-Joon Ahn, editors, *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*, pages 179–189. ACM, 2015.

[62] National Institute of Standards and Technology (NIST). Digital Signature Standard (DSS), July 2013. https://doi.org/10.6028/NIST.FIPS.186-4.

[63] Payment Card Industry (PCI). PIN Security Requirements Version 2.0, December 2014.

[64] Birgit Pfitzmann and Ahmad-Reza Sadeghi. Anonymous fingerprinting with direct non-repudiation. In Tatsuaki Okamoto, editor, *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*, volume 1976 of *Lecture Notes in Computer Science*, pages 401–414. Springer, 2000.

[65] Le Trieu Phong, Lihua Wang, Yoshinori Aono, Manh Ha Nguyen, and Xavier Boyen. Proxy re-encryption schemes with key privacy from LWE. *IACR Cryptology ePrint Archive*, 2016:327, 2016.

[66] Henrik Pierrou. Emulation tool for credit card interface validation and authorization. Master's thesis, KTH Royal Institute of Technology, 2005.

[67] Yuriy Polyakov, Kurt Rohloff, Gyana Sahu, and Vinod Vaikuntanathan. Fast proxy re-encryption for publish/-subscribe systems. *ACM Trans. Priv. Secur.*, 20(4):14:1–14:31, 2017.

[68] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad S. Wahby. Pairing-Friendly Curves. Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-07, Internet Engineering Task Force, June 2020. Work in Progress.

[69] Nolen Scaife, Christian Peeters, and Patrick Traynor. Fear the reaper: Characterization and fast detection of card skimmers. In William Enck and Adrienne Porter Felt, editors, *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*, pages 1–14. USENIX Association, 2018.

[70] Graham Steel. Formal analysis of PIN block attacks. *Theor. Comput. Sci.*, 367(1-2):257–270, 2006.

[71] Visa. Chip Advisory 20, Visa Recommended Practices for EMV Chip Implementation in the U.S., 11 July 2012.

[72] Jian Weng, Robert H. Deng, Shengli Liu, Kefei Chen, Junzuo Lai, and Xu An Wang. Chosen-ciphertext secure proxy re-encryption without pairings. *IACR Cryptol. ePrint Arch.*, 2008:509, 2008.

A Derived Ciphertexts

Canetti and Hohenberger's [27] definition of derived ciphertexts differs from ours in the fourth condition. Our *external derivation* condition states that (i,c) is derived from (i^*,c^*) if

> the adversary queries $\mathcal{O}_{\mathsf{RKG}}(i^*,i)$, or equivalently $\mathcal{O}_{\mathsf{RKG}}(i,i^*)$ in the bidirectional case, to get a re-encryption key from i^* to i, $rk_{i^*\to i}$, and $c = \mathsf{RE}(rk_{i^*\to i},c^*)$.

In contrast, Canetti and Hohenberger's external derivation condition states that (i,c) is derived from (i^*,c^*) if

> the adversary queries $\mathcal{O}_{\mathsf{RKG}}(i,j)$ or $\mathcal{O}_{\mathsf{RKG}}(j,i)$ to get a re-encryption key from i to j, and $\mathcal{O}_{\mathsf{D}}(j,c)$ results in a challenge plaintext m_0 or m_1, then (j,c) is a derivative of (i,c') for all c'.

Canetti and Hohenberger's definition is used to deal with the fact that re-encryption could be randomized, and if the adversary has generated a re-encryption key, then it could use any randomness it wanted to, to re-encrypt. Using Canetti and Hohenberger's external derivation condition, if an adversary makes a challenge query $O_{\mathsf{Chall}}(i,m_0,m_1) = c$ and requests a re-encryption key from i to j, then any re-encryption of c from i to j cannot be entered into a decryption query involving j. Although this prevents trivial wins, it also prevents the adversary from making encryption queries $O_{\mathsf{E}}(i,m_0)$ or $O_{\mathsf{E}}(i,m_1)$ and then requesting the decryption of those ciphertexts, which should be a harmless operation since $O_{\mathsf{Chall}}(i,m_0,m_1)$ should use independent randomness anyway. It is also for this reason that Canetti and Hohenberger's CCA definition is not comparable with Cohen's HRA definition [34].

Since we are mainly interested in analyzing schemes with deterministic re-encryption, we adopt a less restrictive external condition. Deterministic re-encryption means that each ciphertext has only one re-encryption per re-encryption key, therefore it is possible to track the re-encryptions of the challenge ciphertext, and not exclude harmless encryptions of the challenge plaintexts.

B Unidirectional PRE based transaction

Here we detail a payment transaction in the case of unidirectional PRE:

1. Using the pk_{gw} provided by the merchant bank/PoS partner, the PoS application invokes PRE.E on the user-supplied PIN and sends the encrypted pin block to the payment gateway.
2. The payment gateway uses his key $rk_{gw\to mb}$ to invoke PRE.RE, and send the re-encrypted PIN to the merchant bank.
3. The merchant bank invokes PRE.RE with the key $rk_{mb\to np}$, to reencrypt towards the corresponding network processor and sends on the result.
4. The network processor determines the appropriate the consumer bank that must authorize the transaction. It then looks up the corresponding re-encryption key and invokes PRE.RE with the key $rk_{np\to cb}$, and sends the re-encrypted PIN block to that bank.
5. The consumer bank's HSM uses the private key sk_{cb} to invoke PRE.D (producing the cleartext PIN within HSM), and authenticates the PIN based on some additional secret data needed for PIN verification.

C Surveying Known PRE Schemes

We consider four properties when evaluating PRE schemes (see Section 4 for definitions of the terms used):

1. whether they are single-hop or multi-hop; our setting requires PRE schemes which are multi-hop;
2. what security level they offer; known constructions are either CPA, HRA, or CH-CCA secure;
3. whether they offer unidirectional or bidirectional re-encryption; although both can be used in the payments setting, unidirectional PRE schemes simplify the key setup process;
4. their efficiency; using PRE should not significantly increase transaction latency compared to HSMs, hence ciphertext size should be small and computationally demanding operations avoided.

As single-hop schemes cannot be used in our setting, we rule out constructions by Atienese et al. [14, 15], Libert and Vergnaud [55, 56], Chow et al. [32], Kirshanova [52], Phong et al. [65], Dutta et al. [39], Hanaoka et al. [48], and Weng et al. [37, 72].

From the multi-hop schemes, we cannot use those which are only CPA secure, ruling out the constructions by Nuñez, Agudo, and López [61], Blaze, Bleumer, and Strauss [23], Polyakov et al. [67], and Borcea et al. [26]. Fan and Liu (FL) [41, 42] use lattices to build a unidirectional scheme which is CCA secure, and a CPA-secure multi-hop scheme.

Some have proposed CH-CCA-secure, multi-hop schemes. Lai et al. [54] construct one based on indistinguishability obfuscation, a primitive which is not practical, requiring potentially years to evaluate [18] and whose cryptanalysis is nascent, making it unsuitable to deploy.

Canetti and Hohenberger (CH) [27] use pairings to build a CH-CCA-secure, multi-hop, bidirectional scheme. They do not instantiate their scheme, hence we estimate its efficiency by instantiating its underlying components. The scheme uses a pairing-friendly curve, a signature scheme, and a hash function. Let $e : G_1 \times G_2 \to G_T$ denote the pairing, then the ciphertext consists of the verification key of the signature scheme, two G_1 elements, one G_2 element, and one G_T element.

As signature scheme we take ECDSA [62] with 32 byte public keys.

Using the 128-bit secure curves recommended by Sakemi et al. [68], we calculate that ciphertext size with the BN462 curve is 956 bytes, and with the BLS12-381 curve it is 800 bytes. If using supersingular curves, we need G_T to be at least 3072 bits to ensure the finite-field discrete-log problem is difficult [35] per NIST recommendations [19]; with an embedding degree of two, this results in a ciphertext of more than 990 bytes. Given the fact that a typical transaction is roughly 200-300 bytes long, using CH would at least double the transaction size, thereby increasing network costs.

The pairing operation with pairing-friendly elliptic curves is at least an order of magnitude slower than exponentiation on normal elliptic curves [24,25], which would make PIN translation significantly slower than an HSM. Overall, the combination of increased transaction size and computation slowdown would increase latency and bandwidth requirements, thereby making CH an unattractive solution in practice.

D Additional Cryptographic Definitions

Definition 13 (Unidirectional Proxy Re-encryption Scheme)**.** *A* Unidirectional Proxy Re-Encryption scheme *is a public-key encryption scheme with re-key generation* RKG *and re-encryption* RE *functionalities. Re-key generation has the following interface:*

$\mathsf{RKG}(sk_i, pk_i, pk_j) \to rk_{i\to j}$: *on input a source key pair* (sk_i, pk_i) *and destination public key* pk_j, RKG *outputs a re-encryption key* $rk_{i\to j}$.

Re-encryption has the following interface:

$\mathsf{RE}(rk_{i\to j}, c) \to c'$ or $\perp$: *on input a re-encryption key* $rk_{i\to j}$ *and a ciphertext c,* RE *outputs a transformed ciphertext* c' *or error symbol,* $\perp$.

Definition 14 (KEM Correctness)**.** *For all* (pk, sk) *output by* KG*, with probability one it holds that*

$$(c,k) = \mathsf{Encaps}(pk) \tag{11}$$

$$k = \mathsf{Decaps}(sk, c)\,. \tag{12}$$

Definition 15 (ℓ-hop KREM Correctness)**.** *A KREM scheme is* ℓ*-hop correct if it is correct as a KEM and for all possible* (pk_i, sk_i) *generated by* KG *for* $i \le i \le \ell+1$*, for all* $rk_{i\to i+1}$ *generated by* RKG *for* $1 \le i \le \ell$*, with probability one it holds that:*

$$(c_1, k) = \mathsf{Encaps}(pk_1) \tag{13}$$

$$c_{i+1} = \mathsf{ReEncaps}(rk_{i\to i+1}, c_i) \qquad 1 \le i \le \ell \tag{14}$$

$$k = \mathsf{Decaps}(sk_{\ell+1}, c_{\ell+1})\,. \tag{15}$$

Definition 16 (Multi-hop KREM Correctness)**.** *A KREM scheme is multi-hop correct if it is* ℓ*-hop correct for all* $\ell \ge 1$.

Definition 17 (DEM Correctness)**.** *A DEM is correct if for all k output by* KG *and all messages m, it holds with probability one that*

$$\mathsf{D}(k, \mathsf{E}(k,m)) = m\,. \tag{16}$$

Definition 18 (KEM CCA Advantage)**.** *Consider a KEM* $(\mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps})$*, then the CCA advantage of an adversary* **A** *is*

$$\mathbf{Adv}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}\left(pk, c, k, \mathsf{Decaps}_{sk}\,;\, pk, c, k', \mathsf{Decaps}_{sk}\right)\,, \tag{17}$$

where $(pk, sk) \leftarrow \mathsf{KG}$*,* $(c,k) \leftarrow \mathsf{Encaps}(pk)$*,* k' *is chosen uniformly from all strings of length* $|k|$*, and* **A** *may not input c to its* Decaps_{sk} *oracle.*

Figure 8: KREM-hrCCA Game Oracles

```
function $O_{\mathsf{RKG}}(i,j)$                          ▷ ReKey Generation
    if $\{i,j\} \not\subseteq H$ or $\{i,j\} \not\subseteq C$ then return $\perp$
    return $rk_{i\to j}$
function $O_{\mathsf{Chall}}(i)$                                ▷ Challenge
    if $i \notin H$ then return $\perp$
    $(c, K_0) \leftarrow \mathsf{KREM.Encaps}(pk_i)$
    $K_1 \xleftarrow{\$} \{0,1\}^{|K_0|}$
    return $(c, K_b)$
function $O_{\mathsf{Encaps}}(i)$                           ▷ Encapsulation
    return $\mathsf{KREM.Encaps}(pk_i)$
function $O_{\mathsf{ReEncaps}}(i,j,c)$                    ▷ HRA Re-encaps.
    if $c \notin D^* \cup D$, or $c \in D^*$ and $j \in C$ then
        return $\perp$
    return $\mathsf{KREM.ReEncaps}(rk_{i\to j}, c)$
function $O_{\mathsf{Decaps}}(i,c)$                         ▷ Decapsulation
    if $c \in D^*$ then return $\perp$
    return $\mathsf{KREM.Decaps}(sk_i, c)$
```

Definition 19 (KREM-hrCCA Advantage)**.** *Consider a key re-encapsulation mechanism* $\mathsf{KREM} := (\mathsf{KG}, \mathsf{Encaps}, \mathsf{Decaps}, \mathsf{RKG}, \mathsf{ReEncaps})$*. Let* **A** *be an adversary playing game* G_b *denoting an execution of the KREM-hrCCA game defined below, with bit b set to either* 0 *or* 1*. Then* **A***'s* KREM-hrCCA advantage *is defined as*

$$\mathbf{Adv}_{KREM}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}(G_0\,;\, G_1)\,. \tag{18}$$

The KREM-hrCCA security games G_b *(for* $b \in \{0,1\}$*) are played in two phases as follows:*

Setup*: Run the KREM setup and give the public parameters generated to the adversary. The adversary returns n, H, and C, representing the total number of parties, the set of honest parties, and corrupt parties, respectively. Generate n keypairs* (pk_i, sk_i) *with* KG*, and divide the indices among the sets H and C. For all* $i \in H$ *give the adversary* pk_i *and for all* $i \in C$ *give the adversary* (pk_i, sk_i).

Attack*: For all* $i, j \le n$*,* $i \ne j$*, compute the re-encryption key* $rk_{i\to j}$ *using* RKG*. We let* D^* *denote the set of ciphertexts which are derived from the challenge ciphertext (cf. Definition 6);* $D^* = \emptyset$ *before the challenge oracle is queried. Similarly, we let D denote the set of ciphertexts which are derived from any encapsulation oracle output. The adversary* **A** *is given access to the oracles as specified in Figure 8, where* **A** *may query* O_{Chall} *at most once. At the end of the attack phase the adversary outputs a guess* b'.

A family of function is a map $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$. If $K \in \mathcal{K}$, then we denote the corresponding instance as $E_K(\cdot)$ (and inverse function $D_K(\cdot)$). We assume that $|E_K(M)| = l(|M|)$ for some length function $l(\cdot)$. Let $\mathrm{Rand}(\mathcal{M}, l)$ be the set of all functions mapping strings in $\mathcal{M}$ to random l-bit strings. Let $\mathrm{Perm}(\mathcal{M})$ be the set of all length-preserving, one-to-one and onto functions on $\mathcal{M}$.

Definition 20 (Pseudorandom Function/Permutation (PRF/PRP) Family)**.** *Let* **A** *be a distinguisher with access to an oracle. If $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ is a function family with length function l, the advantage of the distinguisher is defined as:*

$$\mathbf{Adv}_{\mathsf{prf}}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}(E_K\,;f)\ ,$$

$$\mathbf{Adv}_{\mathsf{prp}}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}(E_K\,;\pi)\ ,$$

where $K \overset{\$}{\leftarrow} \mathcal{K}$, $f \overset{\$}{\leftarrow} \mathrm{Rand}(\mathcal{M}, l)$ and $\pi \overset{\$}{\leftarrow} \mathrm{Perm}(\mathcal{M})$.

Definition 21 (Strong Pseudorandom Permutation (SPRP) Family)**.** *Let* **A** *be a distinguisher with access to an oracle. If $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ is a PRP with length function l, the advantage of the distinguisher is defined as:*

$$\mathbf{Adv}_{\mathsf{sprp}}(\mathbf{A}) := \underset{\mathbf{A}}{\Delta}\left(E_K, D_K\,;\pi, \pi^{-1}\right)\ ,$$

where $K \overset{\$}{\leftarrow} \mathcal{K}$ and $\pi \overset{\$}{\leftarrow} \mathrm{Perm}(\mathcal{M})$.

E Proof of Theorem 2

Recall that an adversary **B**'s advantage in playing the gap-SDDH-RO game is

$$\underset{\mathbf{B}}{\Delta}\left(F, F_v, g^u, g^v, g^{1/v}, F(g^{uv})\,; F, F_v, g^u, g^v, g^{1/v}, W\right)\ . \qquad (19)$$

We construct gap-SDDH-RO adversary **B** as follows.

1. **B** is given $g^u, g^v, g^{1/v}, W, F_v$, where W is either $F(g^{uv})$ or chosen uniformly at random.

2. **B** runs **A** and the setup procedure for the KREM hrCCA game as follows. **A** gives **B** two disjoint subsets of $\{1, \ldots, n\}$, H and C, representing the honest and corrupt users, respectively. For $i \in C$, **B** generates independent keys (pk_i, sk_i) using $\mathsf{KREM.KG}$. For $i \in H$, **B** generates uniform random values r_i. The adversary **B** gives **A** $pk_i = (g^v)^{r_i} = g^{vr_i}$ for $i \in H$, and (pk_i, sk_i) for $i \in C$.

3. **B** initiates **A**'s attack phase, and responds to **A** oracle queries as specified in Figure 9; if **A** makes a query to the random oracle, then **B** just forwards the query to its own random oracle F; **B** uses variables which are initialized as follows:

   (a) the table of rekeys computed for **A**, $RK[\cdot,\cdot]$, is initialized to $\perp$ for all i, j,

   (b) the ChallC variable, which keeps track of the output of **A**'s single challenge query, is initialized to $\perp$, and

   (c) the mapping $R[\cdot,\cdot]$, is initialized to $\perp$ everywhere; it represents the mapping from tuples (i, c) with $i \in \{1, \ldots, n\}$ and c a ciphertext to tuples (i', r), where i' is the identity of the user that did the initial encapsulation and r the random value used during the initial encapsulation, .

We argue below that **B** simulates **A**'s KREM-hrCCA game perfectly. Since the output of the challenge query includes W, which is either $F(g^{uv})$ or uniform random, if **A** wins — meaning it correctly guesses the bit b — then **B** wins its game.

The adversary **B** generates keys for the corrupt users as done by the KREM-hrCCA game. The honest keys that **B** generates are of the form g^{vr_i}, which are distributed uniformly since the r_i are chosen uniformly at random. Hence **B**'s setup procedure for **A** operates as the KREM-hrCCA's setup procedure.

We argue that each oracle that **B** simulates behaves as it would in the KREM-hrCCA game, assuming the secret keys of the honest users' keys are vr_i:

1. $\mathcal{O}_{\mathsf{RKG}}(i, j)$: if both (i, j) are corrupt then **B** operates normally; if both (i, j) are honest, then their rekey is $(vr_i)/(vr_j) = r_i/r_j$, hence **B** returns the correct rekey.

2. $\mathcal{O}_{\mathsf{Chall}}(i)$: since u is chosen uniformly at random, u/r_i is distributed uniformly at random as well. Setting $r := u/r_i$, and interpreting r to be the randomness normally chosen during a challenge query, we see that g^r is returned along with W. The value W is either uniform random, or

$$F(g^{uv}) = F\left((g^{vr_i})^{u/r_i}\right) = F\left(pk_i^r\right)\ . \qquad (20)$$

3. $\mathcal{O}_{\mathsf{Encaps}}(i)$: encapsulation is the same as in the KREM-hrCCA game.

4. $\mathcal{O}_{\mathsf{ReEncaps}}(i, j, c)$: re-encapsulation starts by running the isDerived procedure, to determine whether (i, c) is derived from either the challenge ciphertext or some prior $\mathcal{O}_{\mathsf{Encaps}}$ output. If isDerived works as intended, then it should output $\perp$ if (i, c) is not derived from either the challenge ciphertext or a prior $\mathcal{O}_{\mathsf{Encaps}}$ output, and otherwise isDerived outputs (i^*, r), where i is the $\mathcal{O}_{\mathsf{Chall}}$ or $\mathcal{O}_{\mathsf{Encaps}}$ input from which (i, c) is derived, and r is the randomness used during that $\mathcal{O}_{\mathsf{Chall}}$ or $\mathcal{O}_{\mathsf{Encaps}}$ query.

   Assume isDerived works as intended. Then if isDerived's output is $\perp$, (i, c) is neither derived from the challenge query, nor from an encapsulation query, and so $\mathcal{O}_{\mathsf{ReEncaps}}$ should output $\perp$. Similarly, if $j \in C$ and i^* is the input to the challenge query, i.e. $i^* = \mathrm{ChallC}[1]$, (i, j, c) is an invalid input to $\mathcal{O}_{\mathsf{ReEncaps}}$ and $\perp$ is returned.

   Otherwise, re-encapsulation works as in the actual KREM-hrCCA game. If both i, j are either corrupt or honest, **B** can perform the proper re-encapsulation since it has the correct re-encapsulation keys. If i is corrupt, and j is honest, then **B** uses the recovered randomness r to return

$$(g^{1/v})^{sk_i r/r_j} = (g^r)^{sk_i/(vr_j)} = (g^r)^{sk_i/sk_j} = (g^r)^{rk_{i\to j}}\ . \qquad (21)$$

FIGURE 9: THE ORACLES SIMULATED BY B

```
1:  function $O_{\mathsf{RKG}}(i,j)$                                ▷ ReKey Generation
2:      if $\{i,j\} \not\subseteq H$ or $\{i,j\} \not\subseteq C$ then return $\perp$
3:      else if $\{i,j\} \subseteq C$ then
4:          $rk \leftarrow \mathsf{KREM.RKG}((pk_i, sk_i), (pk_j, sk_j))$
5:      else if $\{i,j\} \subseteq H$ then
6:          $rk \leftarrow r_i / r_j$
7:      $RK[i,j] \leftarrow rk$
8:      $RK[j,i] \leftarrow 1/rk$
9:      return $rk$
10: function $O_{\mathsf{Chall}}(i)$                                   ▷ Challenge
11:     if $i \notin H$ then return $\perp$
12:     $\mathrm{ChallC} \leftarrow (i, (g^u)^{1/r_i})$; $R[i, (g^u)^{1/r_i}] \leftarrow (i, *)$
13:     return $((g^u)^{1/r_i}, W)$
14: function $O_{\mathsf{Encaps}}(i)$                                  ▷ Encapsulation
15:     $r \overset{\$}{\leftarrow} \mathbb{Z}_q^*$; $c \leftarrow g^r$; $R[i,c] \leftarrow (i,r)$
16:     return $(c, F(pk_i^r))$
17: function $O_{\mathsf{ReEncaps}}(i,j,c)$                         ▷ Re-encapsulation
18:     $x \leftarrow \textsc{isDerived}(i,c)$
19:     if $x = \perp$ then return $\perp$
20:     $(i^*, r) \leftarrow x$
21:     if $j \in C$ and $i^* = \mathrm{ChallC}[1]$ then return $\perp$
22:     if $\{i,j\} \subseteq C$ then
23:         $y \leftarrow \mathsf{KREM.ReEncaps}(sk_i / sk_j, c)$
24:     else if $\{i,j\} \subseteq H$ then
25:         $y \leftarrow \mathsf{KREM.ReEncaps}(r_i / r_j, c)$
26:     else if $i \in C, j \in H$ then
27:         $y \leftarrow (g^{1/v})^{sk_{i*} r / r_j}$
28:     else if $i \in H, j \in C$ then
29:         $y \leftarrow (g^v)^{r r_{i*} / sk_j}$
30:     $R[j,y] = R[i,c]$
31:     return $y$
32: function $O_{\mathsf{Decaps}}(i,c)$                                ▷ Decapsulation
33:     $x \leftarrow \textsc{isDerived}(i,c)$
34:     if $x = \perp$ then return $\perp$
35:     $(i^*, r) \leftarrow x$
36:     if $i^* = \mathrm{ChallC}[1]$ then return $\perp$
37:     if $i \in H$ then
38:         return $F_v(c^{r_i})$
39:     return $\mathsf{KREM.Decaps}(sk_i, c)$
40: function $\textsc{isDerived}(i,c)$
41:     if $R[i,c] \neq \perp$ then return $R[i,c]$
42:     for all $j$ where $RK[i,j] \neq \perp$ do
43:         if $R[j, c^{RK[i,j]}] \neq \perp$ then
44:             $R[i,c] = R[j, c^{RK[i,j]}]$
45:     return $R[i,c]$
```

The adversary **B** performs a similar computation if i is honest and j is corrupt.

We argue that isDerived works as intended. The isDerived procedure outputs either $\perp$ or a tuple. If isDerived outputs $\perp$, then $R[i,c] = \perp$. If $R[i,c] = \perp$, then

(a) there was no prior $O_{\mathsf{Chall}}(i)$ or $O_{\mathsf{Encaps}}(i)$ query with c as the left half of the output, and

(b) there was no prior $O_{\mathsf{ReEncaps}}(i', i, c') = c$ query, and

(c) either $RK[i,j] = \perp$ for all j, meaning **A** never queried $O_{\mathsf{RKG}}(j,i)$ or $O_{\mathsf{RKG}}(i,j)$, or

there is some j such that $RK[i,j] \neq \perp$, but $R[j, c^{RK[i,j]}] = \perp$.

Continuing the same reasoning by induction on $(j, c^{RK[i,j]})$, we conclude that (i,c) could not have been derived from either a challenge or an encapsulation query.

Conversely, say that isDerived outputs $(i^*, r) = R[i,c] \neq \perp$. Then $R[i,c]$ could have been set during an $O_{\mathsf{Chall}}(i)$ or $O_{\mathsf{Encaps}}(i)$ query, or it could have been set at line 30 of Figure 9 during a $O_{\mathsf{ReEncaps}}(i,j,c)$. In those cases (i,c) is the result of a challenge or encapsulation query, or re-encapsulation of another ciphertext (i^*, c^*) for which $R[i^*, c^*] \neq \perp$. The value $R[i,c]$ also could have been set during the loop in the isDerived call, which searches for all rekeys for which **A** has queried O_{RKG}, and (i,c) is a re-encapsulation of some other ciphertext for which R is not $\perp$. Hence we conclude that if (i^*, r) is output by isDerived, then (i,c) is derived from either an encapsulation query or a challenge query. Furthermore, following the code, we see that i^* is the input to the encapsulation or challenge query, and r is the randomness used.

5. $O_{\mathsf{Decaps}}(i,c)$: following the discussion above, we know that isDerived works as intended. Therefore **B** detects whether (i,c) is derived from the challenge ciphertext and returns $\perp$ if that is the case. Otherwise, if i is corrupt, it can de-encapsulate as normal, and if i is honest, it can de-encapsulate using F_v. Note that (i,c) does not need to be derived from an encapsulation query during a decapsulation query.

F Proof of Theorem 3

In this section we prove that the EPB defined in Figure 2 and 3 is a secure DEM. Our proof relies on the results of Bellare and Rogaway [21] which formalize the Encode-then-Encipher paradigm. We begin be recalling two definitions from Bellare and Rogaway's paper which define properties of encoding schemes.

An encoding scheme $\mathsf{ES} = (\mathsf{encode}, \mathsf{decode})$ maps between a message space $\mathcal{M}$ and an encoded message space $\mathcal{M}^*$. The function decode on input M^* will either return $M \in \mathcal{M}$ or the symbol $\perp$ if an invalid message is submitted. We require that for an encoding scheme to be correct $\mathsf{decode}(\mathsf{encode}(M)) = M$ for any $M \in \mathcal{M}$.

Definition 22 (Rare-Collision Encodings)**.** *An encoding scheme* $\mathsf{ES} = (\mathsf{encode}, \mathsf{decode})$ *is* ε*-colliding if for any adversary* $\mathbf{A}$ *making* q *queries, the probability that two queries create the same valid response is* $\varepsilon(q)$.

Definition 23 (Sparse Encodings)**.** *An encoding scheme* $\mathsf{ES} = (\mathsf{encode}, \mathsf{decode})$ *is* δ*-dense if for all* $n \in \mathbb{N}$,

$$\mathbf{P}\left[M^* \xleftarrow{\$} \{0,1\}^n : \mathsf{decode}(M^*) \in \mathcal{M}\right] \leq \delta.$$

Proof. In order to prove that the symmetric encryption scheme SE of Figure 3 is a secure DEM we must show it satisfies Definition 11, recall that the adversary is only permitted one query to the encryption oracle. By the result of Bellare and Namprempre [20, Theorem 3.2] we know that a scheme which is both IND-CPA and INT-CTXT secure is a CCA secure, we refer the reader to the original paper of Bellare and Namprempre for the definitions and proof.

Next we use the results of Bellare and Rogaway [21, Theorems 4.1 and 4.2]. First let us consider the IND-CPA security. Here by Bellare and Rogaway's first result we have that:

$$\mathbf{Adv}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(\mathbf{A}') \leq \mathbf{Adv}_{\mathsf{prf}}(\mathbf{B}) + \varepsilon(q),$$

since for a DEM we only consider $q = 1$, i.e. only a single oracle call is permitted, we have that:

$$\mathbf{Adv}_{\mathsf{SE}}^{\mathsf{ind-cpa}}(\mathbf{A}') \leq \mathbf{Adv}_{\mathsf{prf}}(\mathbf{B}) + \varepsilon(1) = \mathbf{Adv}_{\mathsf{prf}}(\mathbf{B}).$$

By making only one query there is never any chance of a collision. Thus if F is a PRF we achieve IND-CPA security.

Next consider the INT-CTXT security. Here by Bellare and Rogaway's second result we have that for $q \leq \frac{1}{2\delta}$:

$$\mathbf{Adv}_{\mathsf{SE}}^{\mathsf{int-ctxt}}(\mathbf{A}'') \leq \mathbf{Adv}_{\mathsf{sprp}}(\mathbf{C}) + 2\delta.$$

It then remains to bound δ for the ISO-0 and ISO-3 encoding schemes as defined in Figure 2. Note that in Bellare and Rogaway's original definition of an encoding scheme there is only one input to the encode function. For the definition of ISO-0 and ISO-3 we take two inputs, the PAN and the PIN. However, we can view the PAN as associated data since it is also sent separately to encoded/encrypted PIN and is required as an additional input to the decode function. This allows us to view the PAN as a fixed masked applied by the encoding scheme. As a result we directly apply the result of Bellare and Rogaway by purely focusing on the δ-density of the encoding of the plaintext PIN field.

For ISO-0 and ISO-3 (due to our application we assume a PIN length of 4 is mandated), the probability that a randomly chosen string has a valid control field is $1/2^4$, a valid PIN length is $1/2^4$ and a valid PIN digit is $10/2^4$. For ISO-0 the probably of having a valid fill digit is $1/2^4$ and for ISO-3 it is $6/2^4$. We therefore obtain the following bounds:

$$\delta_{\mathrm{ISO-0}} = \frac{1}{2^4} \cdot \frac{1}{2^4} \cdot \left(\frac{10}{2^4}\right)^4 \cdot \left(\frac{1}{2^4}\right)^{10} = \frac{5^4}{2^{60}} \leq 2^{-48},$$

$$\delta_{\mathrm{ISO-3}} = \frac{1}{2^4} \cdot \frac{1}{2^4} \cdot \left(\frac{10}{2^4}\right)^4 \cdot \left(\frac{6}{2^4}\right)^{10} = \frac{5^4 \cdot 3}{2^{59}} \leq 2^{-45}.$$

Therefore SE is secure in the sense of INT-CTXT assuming that F is a SPRP.

By applying the result of Bellare and Namprempre [20, Theorem 3.2] we achieve DEM-CCA security. □