

Payment with Dispute Resolution: A Protocol For Reimbursing Frauds' Victims

Abstract. An “Authorised Push Payment” (APP) fraud refers to the case where fraudsters deceive a victim to make payments to bank accounts controlled by them. The total amount of money stolen via APP frauds is swiftly growing. Although regulators have provided guidelines to improve victims’ protection, the guidelines are vague and the victims are not receiving sufficient protection. To *facilitate victims’ reimbursement*, in this work, we propose a protocol called “Payment with Dispute Resolution” (PwDR) and formally define it. The protocol lets an honest victim prove its innocence to a third-party dispute resolver while preserving the protocol participants’ *privacy*. It makes black-box use of a standard online banking system. We evaluate its asymptotic cost and runtime via a prototype implementation. Our evaluation indicates that the protocol is efficient. It imposes only $O(1)$ overheads to the customer and bank. Moreover, it takes a dispute resolver only 0.09 milliseconds to settle a dispute between the two parties.

1 Introduction

An “Authorised Push Payment” (APP) fraud is a type of cyber-crime where a fraudster tricks a victim into making an authorised online payment into an account controlled by the fraudster. It is defined by the “Financial Conduct Authority” (FCA) as “*a transfer of funds by person A to person B, other than a transfer initiated by or through person B, where: (1) A intended to transfer the funds to a person other than B but was instead deceived into transferring the funds to B; or (2) A transferred funds to B for what they believed were legitimate purposes but which were in fact fraudulent*” [8]. The APP fraud has various variants, such as romance, investment, CEO, or invoice frauds [42]. The total amount of money lost to APP frauds is substantial. According to a report produced by “UK Finance”, only in the first half of 2021, a total of £355 million was lost to APP frauds, which has increased by 71% compared to losses reported in the same period in 2020 [41]; for the first time, the amount of money stolen via APP frauds overtook even card fraud losses [41]. The UK Finance report suggests that *online* payment is the type of payment method the victims used to make the authorised push payment in 98% of cases. The true cost of APP frauds often extends beyond the immediate financial loss, to additional fees imposed by investigating the event, remediation of internal banks’ processes, and dealing with the emotional fallout of the fraud. APP fraud is a *global* phenomenon. According to the FBI’s report, victims of APP frauds reported to it at least a total of \$419 million losses¹, in 2020 [18]. Recently, Interpol warned its member countries about a variant of APP fraud called investment fraud via dating software [40], also known as “pig butchering” in China [45]. According to Europol’s notice, at least five variants of APP fraud are among the seven most common types of online financial fraud [36].

Although the amount of money lost via APP frauds and the number of cases have been significantly increasing, the victims are not receiving enough protection. In the first half of 2021, only 42% of the stolen funds returned to victims of APP frauds in the UK [41]. Previous years had even lower rates than that. Despite the UK’s financial regulators (unlike US and EU) having provided specific reimbursement regulations (e.g., CRM code) to financial institutes to improve APP frauds victims’ protection, these regulations are ambiguous and open to interpretation. Furthermore, there exists no transparent and uniform mechanism via which honest victims can *prove* their innocence. Currently, each bank uses its own ad-hoc (manual) dispute resolution process which is not transparent to customers, regulators, or consumer protection organisations and is not uniform among all banks and even among those organisations that settle disputes between banks and customers. The APP frauds are not specific to the regular online banking systems, they will eventually

¹ We excluded the losses to “romance” and “government impersonation”, accounting for \$710 million, from the above estimation, as the FBI’s report defines them broadly, that could cover also frauds unrelated to APP ones. Thus, it is likely that the total sum of losses due to APP frauds reported to the FBI is higher than our estimation.

find their way into other payment systems, such as cryptocurrency. To date, the APP fraud problem has been overlooked by the information security and cryptography research communities.

In this work, to facilitate the compensation of APP frauds’ victims for their loss, we propose a protocol called “Payment with Dispute Resolution” (PwDR), present its formal definition, and prove the protocol’s security. The PwDR lets an honest victim (of an APP fraud) independently prove its innocence to a (potentially semi-honest) third-party dispute resolver, in order to be reimbursed. We identify three crucial properties that such a scheme should possess; namely, (a) security against a malicious victim: a malicious victim who is not qualified for the reimbursement should not be reimbursed, (b) security against a malicious bank: a malicious bank cannot disqualify an honest victim from being reimbursed, and (c) privacy: the customer’s and bank’s messages remain confidential from non-participants of the scheme, and a party which resolves dispute learns as little information as possible. The PwDR makes black-box use of a standard online banking system, hence it does not require significant changes to the existing online banking systems. It automates the implementation of the reimbursement regulations where possible, and distributes the role of making more subjective decisions among multiple auditors.

The PwDR offers *transparency* by (i) accurately formalising reimbursements’ conditions: it presents an accurate publicly available formalisation capturing the circumstances under which a customer is reimbursed, (ii) offering traceability: it lets parties’ performance be tracked, and (iii) providing an evidence-based final decision: it requires the reasons leading to the final decision to be accessible and consistent with the reimbursements’ conditions and parties’ actions. It also offers *accountability*, as it is equipped with auditing mechanisms that help identify the party liable for an APP fraud loss. The auditing mechanisms themselves are accompanied by our *lightweight privacy-preserving* threshold voting protocols, which let auditors vote privately without having to worry about being retaliated against, for their votes. Our voting protocols can be of independent interest. We analyse the PwDR’s cost via both asymptotic and runtime evaluation (via a prototype implementation). The evaluation indicates that the protocol is indeed efficient. The customer’s and bank’s complexity are constant, $O(1)$. It only takes 0.09 milliseconds for a dispute resolver to settle a dispute between the two parties. We make the implementation source code publicly available. We hope that our result lays the foundation for future solutions that will protect victims of this concerning type of fraud.

In summary, our contributions are three-fold, we (1) propose an efficient protocol called Payment with Dispute Resolution (PwDR), (2) formally define and prove the PwDR, and (3) analyse both asymptotic and concrete costs of it.

The rest of this paper has been organised as follows. In Section 2, we provide a background on how APP frauds drew regulators’ attention and outline the existing related guidelines. In Section 3, we explain the thread model and the tools we use. In Section 4, we briefly explain the challenges that we need to overcome when designing the PwDR scheme. In Section 1, we provide a formal definition of payment with dispute resolution notion. In Section 6, we give an overview of the PwDR protocol, present a few subroutines (including our threshold voting protocols) along with the detailed PwDR protocol. In Section 7, we formally analyse the security of this protocol. In Section 8, we evaluate the PwDR protocol’s costs, while in Section 9, we provide a set of future research directions. In Section 10, we provide the related work and in Section 11 we conclude the paper. We provide a notation table and more detail about Bloom filters in appendices 3.1 and 3.8, respectively. In Appendix A, we provide the main security theorem and related proof of the first variant of our threshold voting protocol. In Appendix ??, we provide the full protocol of the second variant of our threshold voting scheme. In Appendix B, we provide the latter protocol’s main theorem and proof. In Appendix C, we provide further discussion on these threshold voting protocols.

2 Background

Liability for APP frauds has largely remained with victims who authorise payments. In the UK, there have been efforts to protect the victims. In 2016, the UK’s consumer protection organisation, called “Which?”, submitted a super-complaint to the “Financial Conduct Authority” (FCA) and raised its concerns that despite the APP frauds victims’ rate is growing, the victims do not have enough protection [20]. Since then, the FCA has been collaborating with financial institutions to develop several initiatives that could improve

the response when frauds occur. As a result, the “Contingent Reimbursement Model” (CRM) code [27] was proposed. It lays out a set of requirements and explains under which circumstances customers should be reimbursed by their financial institutions when they fall victim to an APP fraud, e.g., when customers were not grossly negligent, did not ignore effective warnings, or were among a vulnerable group. So far, at least nine firms, comprising nineteen brands (e.g., Barclays, HSBC, Lloyds) signed up to the code.

Although the CRM code is a vital guideline towards protecting the frauds’ victims, some aspects of it are vague, and open to interpretation. For instance, in 2020, the “Financial Ombudsman Service” (that settles complaints between consumers and businesses) had an “overall impression” that firms are applying the CRM code inconsistently and in some cases incorrectly, which resulted in failing to reimburse victims in cases anticipated by this code [39]. As another example, one of the conditions in the CRM code that allows a bank to avoid reimbursing the customer is clause R2(1)(e) which states: *“The Customer has been grossly negligent. For the avoidance of doubt, the provisions of R2(1)(a)-(d) should not be taken to define gross negligence in this context”*. Nevertheless, neither the CRM code nor the “Payment Services Regulations” [38] explicitly define under which circumstances the customer is considered “grossly negligent” in the context of APP frauds. In particular, in the CRM code, the only terms that discuss customers’ misbehaviour are the provisions of R2(1)(a)-(d); however, as stated above, they should be excluded from the definition of the term gross negligence. On the other hand, in the Payment Services Regulations, this term is used three times, i.e., twice in regulation 75 and once in regulation 77. But in all three cases, it is used for frauds related to *unauthorised payments* which are different types of frauds from APP ones. Hence, there is a pressing need for an accurate solution to help and protect APP frauds’ victims.

Unlike the UK that has already recognised APP frauds and developed regulations for that, the US’s financial industry has not distinguished between APP fraud and other types of fraud when labelling their overall fraud losses [1]. Thus, there exist no publicly available accurate reports and regulations concerning APP frauds provided by the financial industry in the US yet. To address the issue, in 2020, the Federal Reserve introduced the “FraudClassifier Model” which classifies payment frauds more granularly [37]. It is yet to be seen how regulations related to APP frauds will be developed and how the payment industry will implement these regulations, in the US. Similarly, in the EU, there exists no specific regulation developed to explicitly capture APP frauds and protect the victims [24,28].

3 Preliminaries

3.1 Notations

We summarise our notations in Table 1.

3.2 Informal Threat Model and Assumptions

A payment with dispute resolution scheme involves six types of parties. Below, we informally explain each type of party’s role. We will provide a formal definition of the scheme in Section 1.

- Customer (\mathcal{C}): it is a regular customer of a bank. We call a customer a victim after it falls victim to an APP fraud. We assume a victim is corrupted by a non-colluding active (or malicious) adversary.
- Bank (\mathcal{B}): it is a regular bank providing online banking. We assume it is corrupted by a non-colluding active adversary. Note that in the real world a bank is not usually an active adversary and cares about its reputation, such as a “rational” or “covert” adversary which is weaker than the active one. However, to ensure our solution offers a strong security guarantee, we assume the adversary is strong too, i.e., active one. We also assume any change to the online banking system’s source code is transparent and can be detected.
- Smart contract (\mathcal{S}): it is a standard smart contract of a public blockchain (e.g., Ethereum). It mainly acts as a tamper-proof public bulletin board to store different parties’ messages.
- Certificate generator (\mathcal{G}): it is a trusted third party (e.g., hospital, registry office) which provides signed digital certificates (e.g., certificate of disability, divorce) to customers.

Table 1: Notation Table.

| Symbol | Description | Symbol | Description |
|---------------------------------------|--|-----------------------------|--|
| $\text{Enc}(\cdot)$ | Encryption algorithm of symmetric key encryption | \mathbf{l} | \mathcal{C} 's payees list |
| $\text{Dec}(\cdot)$ | Decryption algorithm of symmetric key encryption | $\hat{\mathbf{l}}$ | \mathcal{C} 's encoded payees list |
| $\tilde{\text{Enc}}(\cdot)$ | Encryption algorithm of asymmetric key encryption | k_0 | A secret key of PRF |
| $\tilde{\text{Dec}}(\cdot)$ | Decryption algorithm of asymmetric key encryption | f | New payee's detail |
| $\text{keyGen}(\cdot)$ | Key generator algorithm of asymmetric key encryption | in_f | Payment detail |
| $\text{Sig.keyGen}(\cdot)$ | Key generator algorithm of digital signature scheme | \hat{a} | Encoded a |
| $\text{verStat}(\cdot)$ | Algorithm to determine \mathcal{B} 's message status | $\hat{m}_1^{(\mathcal{C})}$ | \mathcal{C} 's encoded update request |
| $\text{checkWarning}(\cdot)$ | Algorithm to check a warning's effectiveness | $\hat{m}_2^{(\mathcal{C})}$ | \mathcal{C} 's encoded payment request |
| $\text{pay}(\cdot)$ | \mathcal{B} 's internal algorithm to transfers money | $\hat{m}_1^{(\mathcal{B})}$ | \mathcal{B} 's encoded warning message |
| $\text{Com}(\cdot)$ | Commitment's commit | $\hat{m}_2^{(\mathcal{B})}$ | \mathcal{B} 's encoded payment message |
| $\text{Ver}(\cdot)$ | Commitment's verify | sk and pk | Secret and public keys |
| $\text{H}(\cdot)$ | Hash function | $sk_{\mathcal{D}}$ | Auditors' secret key |
| $\text{PRF}(\cdot)$ | Pseudorandom function | pp | Public parameter |
| \mathcal{C} | Customer | j | Auditor's index, $1 \leq j \leq n$ |
| \mathcal{B} | Bank | T, T_1, T_2 | Tokens, where $T := (T_1, T_2)$ |
| $\mathcal{D}_1, \dots, \mathcal{D}_n$ | Auditors | w_1 | Output of $\text{verStat}(\cdot)$ |
| \mathcal{DR} | Dispute resolver | (w_2, w_3) | Output of $\text{checkWarning}(\cdot)$ |
| \mathcal{S} | Smart contract | \bar{w}_j | Output of $\text{PVE}(\cdot)$ |
| \mathcal{G} | Certificate generator | v | Output of $\text{FVD}(\cdot)$ |
| SAP | Statement agreement protocol | e | Threshold |
| PVE | Private verdict encoding protocol | $w_{i,j}$ | Auditor's plain verdict |
| FVD | Final verdict decoding protocol | o | Offset |
| GPVE | Generic private verdict encoding protocol | r_j | Pseudorandom value |
| GFVD | Generic final verdict decoding protocol | λ | Security parameter |
| \oplus | XOR operator | μ | Negligible function |
| Δ | Time parameter | in_p | The input of $\text{pay}(\cdot)$ |
| add_I | Address of I | π | Private statement |
| aux, aux' | Auxiliary information | Pr | Probability |
| BF | Bloom filter | ϕ | Null |
| z_1 | \mathcal{C} 's complaint about \mathcal{B} 's message status | n | Total number of auditors |
| z_2 | \mathcal{C} 's complaint about a warning's effectiveness | z | \mathcal{C} 's complaint, where $z := (z_1, z_2, z_3)$ |
| z_3 | \mathcal{C} 's complaint about payment message inconsistency | pwr | payment with dispute resolution notion |
| $g := (g_1, g_2)$ | Commitment values | $PwDR$ | payment with dispute resolution protocol |

- A committee of auditors ($\mathcal{D}_1, \dots, \mathcal{D}_n$): it consists of trusted third-party authorities or regulators (e.g., FCA, financial ombudsman service). They compile complaints and provide their verdicts. If needed, they can access the banking backend software to conduct investigations. We assume they interacted with each other once, to agree on a secret key, \bar{k}_0 , and a pair of keys $(pk_{\mathcal{D}}, sk_{\mathcal{D}})$ of an asymmetric key encryption.
- Dispute resolver (\mathcal{DR}): it is an aggregator of auditors' votes (e.g., public court). Given a collection of votes, it extracts and announces the final verdict. We assume it is corrupted by a non-colluding passive adversary. We assume \mathcal{C} and \mathcal{B} use a secure channel when they send a message directly to \mathcal{DR} .

3.3 Digital Signature

A digital signature is a scheme for verifying the authenticity of digital messages. Below, we restate its formal definition, taken from [23].

Definition 1. A signature scheme involves three algorithms, $\text{Signature} := (\text{Sig.keyGen}, \text{Sig.sign}, \text{Sig.ver})$, that are defined as follows.

- $\text{Sig.keyGen}(1^\lambda) \rightarrow (sk, pk)$. A probabilistic algorithm run by a signer. It takes as input a security parameter. It outputs a key pair (sk, pk) , consisting of secret key sk , and public key pk .

- **Sig.sign**(sk, pk, u) $\rightarrow sig$. An algorithm run by the signer. It takes as input key pair (sk, pk) , and a message u . It outputs a signature sig .
- **Sig.ver**(pk, u, sig) $\rightarrow h \in \{0, 1\}$. A deterministic algorithm run by a verifier. It takes as input public key pk , message u , and signature sig . It checks the signature's validity. If the verification passes, then it outputs 1; otherwise, it outputs 0.

A digital signature scheme should meet the following properties:

- *Correctness*. For every input u it holds that:

$$\Pr \left[\text{Sig.ver}(pk, u, \text{Sig.sign}(sk, pk, u)) = 1 : \text{Sig.keyGen}(1^\lambda) \rightarrow (sk, pk) \right] = 1$$

- *Existential unforgeability under chosen message attacks*. A probabilistic polynomial time (PPT) adversary that obtains pk and has access to a signing oracle for messages of its choice, cannot create a valid pair (u^*, sig^*) for a new message u^* (that was never a query to the signing oracle), except with a small probability, σ . More formally:

$$\Pr \left[u^* \notin Q \wedge \text{Sig.ver}(pk, u^*, sig^*) = 1 : \begin{array}{l} \text{Cer.keyGen}(1^\lambda) \rightarrow (sk, pk) \\ \mathcal{A}^{\text{Sig.sign}(k, \cdot)}(pk) \rightarrow (u^*, sig^*) \end{array} \right] \leq \mu(\lambda)$$

where Q is the set of queries that \mathcal{A} sent to the certificate generator oracle.

3.4 Smart Contract

Cryptocurrencies, such as Bitcoin [29] and Ethereum [43], beyond offering a decentralised currency, support computations on transactions. In this setting, often a certain computation logic is encoded in a computer program, called a “*smart contract*”. Although Bitcoin, the first decentralised cryptocurrency, supports smart contracts, the functionality of Bitcoin’s smart contracts is very limited, due to the use of the underlying programming language that does not support arbitrary tasks. To address this limitation, Ethereum, as a generic smart contract platform, was designed. Thus far, Ethereum has been the most predominant cryptocurrency framework that lets users define arbitrary smart. This framework allows users to create an account with a unique account number or address. Such users are often called external account holders, which can send (or deploy) their contracts to the framework’s blockchain. In this framework, a contract’s code and its related data are held by every node in the blockchain’s network. Ethereum smart contracts are often written in a high-level Turing-complete programming language called “Solidity”. The program execution’s correctness is guaranteed by the security of the underlying blockchain components. To prevent a denial of service attack, the framework requires a transaction creator to pay a fee, called “*gas*”.

3.5 Commitment Scheme

A commitment scheme involves two parties, *sender* and *receiver*, and includes two phases: *commit* and *open*. In the commit phase, the sender commits to a message: x as $\text{Com}(x, r) = \text{Com}_x$, that involves a secret value: $r \xleftarrow{\$} \{0, 1\}^\lambda$. In the end of the commit phase, the commitment Com_x is sent to the receiver. In the open phase, the sender sends the opening $\tilde{x} := (x, r)$ to the receiver who verifies its correctness: $\text{Ver}(\text{Com}_x, \tilde{x}) \stackrel{?}{=} 1$ and accepts if the output is 1. A commitment scheme must satisfy two properties: (a) *hiding*: it is infeasible for an adversary (i.e., the receiver) to learn any information about the committed message x , until the commitment Com_x is opened, and (b) *binding*: it is infeasible for an adversary (i.e., the sender) to open a commitment Com_x to different values $\tilde{x}' := (x', r')$ than that was used in the commit phase, i.e., infeasible to find \tilde{x}' , s.t. $\text{Ver}(\text{Com}_x, \tilde{x}) = \text{Ver}(\text{Com}_x, \tilde{x}') = 1$, where $\tilde{x} \neq \tilde{x}'$. There exist efficient non-interactive commitment schemes both in (a) the standard model, e.g., Pedersen scheme [32], and (b) the random oracle model using the well-known hash-based scheme such that committing is $H(x||r) = \text{Com}_x$ and $\text{Ver}(\text{Com}_x, \tilde{x})$ requires checking: $H(x||r) \stackrel{?}{=} \text{Com}_x$, where $H : \{0, 1\}^* \rightarrow \{0, 1\}^\lambda$ is a collision resistant hash function; i.e., the probability to find x and x' such that $H(x) = H(x')$ is negligible in the security parameter, λ .

3.6 Statement Agreement Protocol

Recently, a scheme called “Statement Agreement Protocol” (SAP) has been proposed in [4]. It lets two mutually distrusted parties, e.g., \mathcal{B} and \mathcal{C} , efficiently agree on a private statement, π . Informally, the SAP satisfies the following four properties: (1) neither party can convince a third-party verifier that it has agreed with its counter-party on a different statement than the one both parties previously agreed on, (2) after they agree on a statement, an honest party can (almost) always prove to the verifier that it has the agreement, (3) the privacy of the statement is preserved (from the public), and (4) after both parties reach an agreement, neither can deny it. The SAP uses a smart contract and commitment scheme. It assumes that each party has a blockchain public address, $adr_{\mathcal{R}}$ (where $\mathcal{R} \in \{\mathcal{B}, \mathcal{C}\}$). Below, we restate the SAP, taken from [4].

1. **Initiate.** $SAP.init(1^\lambda, adr_{\mathcal{B}}, adr_{\mathcal{C}}, \pi)$
The following steps are taken by \mathcal{B} .
 - (a) Deploys a smart contract that explicitly states both parties’ addresses, $adr_{\mathcal{B}}$ and $adr_{\mathcal{C}}$. Let adr_{SAP} be the deployed contract’s address.
 - (b) Picks a random value r , and commits to the statement, $Com(\pi, r) = g_{\mathcal{B}}$.
 - (c) Sends adr_{SAP} and $\tilde{\pi} := (\pi, r)$ to \mathcal{C} , and $g_{\mathcal{B}}$ to the contract.
2. **Agreement.** $SAP.agree(\pi, r, g_{\mathcal{B}}, adr_{\mathcal{B}}, adr_{SAP})$
The following steps are taken by \mathcal{C} .
 - (a) Checks if $g_{\mathcal{B}}$ was sent from $adr_{\mathcal{B}}$, and checks locally $Ver(g_{\mathcal{B}}, \tilde{\pi}) = 1$.
 - (b) If the checks pass, it sets $b = 1$, computes locally $Com(\pi, r) = g_{\mathcal{C}}$, and sends $g_{\mathcal{C}}$ to the contract. Otherwise, it sets $b = 0$ and $g_{\mathcal{C}} = \perp$.
3. **Prove.** For either \mathcal{B} or \mathcal{C} to prove, it sends $\tilde{\pi} := (\pi, r)$ to the smart contract.
4. **Verify.** $SAP.verify(\tilde{\pi}, g_{\mathcal{B}}, g_{\mathcal{C}}, adr_{\mathcal{B}}, adr_{\mathcal{C}})$
The following steps are taken by the smart contract.
 - (a) Ensures $g_{\mathcal{B}}$ and $g_{\mathcal{C}}$ were sent from $adr_{\mathcal{B}}$ and $adr_{\mathcal{C}}$ respectively.
 - (b) Ensures $Ver(g_{\mathcal{B}}, \tilde{\pi}) = Ver(g_{\mathcal{C}}, \tilde{\pi}) = 1$.
 - (c) Outputs $s = 1$, if the checks, in steps 4a and 4b, pass. It outputs $s = 0$, otherwise.

3.7 Pseudorandom Function

Informally, a pseudorandom function is a deterministic function that takes a key of length Λ and an input; and outputs a value indistinguishable from that of a truly random function. In this paper, we use the pseudorandom function: $PRF : \{0, 1\}^\Lambda \times \{0, 1\}^* \rightarrow \mathbb{F}_p$, where p is a large prime number, $|p| = \lambda$, and (Λ, λ) are the security parameters. In practice, a pseudorandom function can be obtained from an efficient block cipher. We refer readers to [23] for a formal definition of a pseudorandom function.

3.8 Bloom Filter

A Bloom filter [10] is a compact data structure for probabilistic efficient elements’ membership checking. A Bloom filter is an array of \bar{m} bits that are initially all set to zero. It represents \bar{n} elements. A Bloom filter comes along with \bar{k} independent hash functions. To insert an element, all the hash values of the element are computed and their corresponding bits in the filter are set to 1. To check an element’s membership, all its hash values are re-computed and checked whether all are set to one in the filter. If all the corresponding bits are one, then the element is probably in the filter; otherwise, it is not. In Bloom filters false positives are possible, i.e., it is possible that an element is not in the set, but the membership query shows that it is.

According to [12], the upper bound of the false positive probability is: $q = \bar{p}^{\bar{k}}(1 + O(\frac{\bar{k}}{\bar{p}}\sqrt{\frac{\ln \bar{m} - \bar{k} \ln \bar{p}}{\bar{m}}}))$, where \bar{p} is the probability that a particular bit in the filter is set to 1 and calculated as: $\bar{p} = 1 - (1 - \frac{1}{\bar{m}})^{\bar{k}\bar{n}}$. The efficiency of a Bloom filter depends on \bar{m} and \bar{k} . The lower bound of \bar{m} is $\bar{n} \log_2 \bar{e} \cdot \log_2 \frac{1}{q}$, where \bar{e} is the base of natural logarithms, while the optimal number of hash functions is $\log_2 \frac{1}{q}$, when \bar{m} is optimal. In this paper, we only use optimal \bar{k} and \bar{m} . In practice, we would like to have a predefined acceptable upper bound on false positive probability, e.g., $q = 2^{-40}$, and can adjust such a parameter. Given q and \bar{n} , we can determine the rest of the parameters. Also, we require that a Bloom filter uses *cryptographic* hash functions.

4 Challenges to Overcome

Our starting point in defining and designing a payment with dispute resolution scheme is the CRM code, as this code (although vaguely) sets out the primary requirements a victim must meet to be reimbursed. To design such a scheme, we need to address several challenges. The rest of this section outlines these challenges.

4.1 Challenge 1: Lack of Transparent Logs

In the current online banking system, during a payment journey, the messages exchanged between customer and bank are usually logged by the bank and are not accessible to the customer without the bank’s collaboration. Even if the bank provides access to the transaction logs, there is no guarantee that the logs have remained intact. Due to the lack of a transparent logging mechanism, a customer or bank can wrongly claim that (a) it has sent a certain message or warning to its counter-party or (b) it has never received a certain message. Thus, it would be hard for an honest party to prove its innocence. To address this challenge, our scheme will use a smart contract to which each party sends its messages.

4.2 Challenge 2: Lack of Effective Warning’s Accurate Definition in Banking

One of the determining factors in the process of allocating liability to an APP fraud victim is following “warning(s)”, according to the CRM code. However, there exists no publicly available study on the effectiveness of banks’ warnings. So, we cannot hold a customer accountable for becoming the fraud’s victim, even if the related warnings are ignored. Also, currently, banks assess whether their own warnings are effective. But, in a fair process, such an assessment is conducted by a third party. To address these challenges, we let a warning’s effectiveness be determined on a case-by-case basis after an APP fraud occurs. The protocol lets a victim challenge a certain warning whose effectiveness will be assessed by a *committee*, i.e., a set of auditors. In this setting, each auditor provides its (encoded) verdict to the smart contract, from which a dispute resolver retrieves all verdicts to learn the final one. The scheme ensures that the final verdict is in the customer’s favour if at least a threshold of the auditors voted so. Thus, unlike the traditional setting where a central party determines a warning’s effectiveness, which is error-prone, we let a collection of auditors determines it.

4.3 Challenge 3: Linking Off-chain Payments with a Smart Contract

Recall that an APP fraud occurs when a payment is made. In the case where a bank sends (to the smart contract) a confirmation of payment message, it is not possible to automatically validate such a claim, as the money transfer occurs outside of the blockchain network. To address this challenge, our scheme lets a customer raise a dispute and report it to the smart contract when it detects an inconsistency. In this case, the above auditors investigate and provide their verdicts to the smart contract. Then, dispute resolver \mathcal{DR} extracts them and announces the final verdict.

4.4 Challenge 4: Preserving Privacy

Although the use of a public logging mechanism is essential in resolving disputes transparently, if it does not use a privacy-preserving mechanism, then parties’ privacy would be violated. To protect the privacy of the bank’s and customers’ messages against the public, our scheme lets them provably agree on encoding-decoding tokens with which they can encode their messages. Later, either party can provide the token to a third party (e.g., \mathcal{D}_i) which checks the token’s correctness, and decodes the messages. To protect the privacy of the committee members’ verdicts from \mathcal{DR} , the scheme ensures that \mathcal{DR} can learn only the final verdict without being able to link a verdict to a specific auditor or even learn the number of yes/1 and no/0 votes. To this end, we develop and use novel threshold voting protocols.

5 Payment with Dispute Resolution Definition

In this section, we outline a formal definition of the payment with dispute resolution (pwdr) notion. Below, we first provide the scheme's syntax. Then, we formally define its correctness and security properties.

Definition 2. A payment with dispute resolution (pwdr) involves six types of entities; namely, bank \mathcal{B} , customer \mathcal{C} , smart contract \mathcal{S} , certificate generator \mathcal{G} , set of auditors $\mathcal{D} : \{\mathcal{D}_1, \dots, \mathcal{D}_n\}$, and dispute resolver \mathcal{DR} . It includes the following algorithms $\text{pwdr} := (\text{keyGen}, \text{bankInit}, \text{customerInit}, \text{genUpdateRequest}, \text{insertNewPayee}, \text{genWarning}, \text{genPaymentRequest}, \text{makePayment}, \text{genComplaint}, \text{verComplaint}, \text{resDispute})$. Below, we define these algorithms.

- $\text{keyGen}(1^\lambda) \rightarrow (sk, pk)$. It is a probabilistic algorithm run independently by \mathcal{G} and one of the auditors, \mathcal{D}_j . It takes as input a security parameter 1^λ . It outputs a pair of secret keys $sk := (sk_{\mathcal{G}}, sk_{\mathcal{D}})$ and public keys $pk := (pk_{\mathcal{G}}, pk_{\mathcal{D}})$. The public key pair, pk , is sent to all participants.
- $\text{bankInit}(1^\lambda) \rightarrow (T, pp, \mathbf{l})$. It is run by \mathcal{B} . It takes as input security parameter 1^λ . It sets private parameters $\tilde{\pi}_1$ and $\tilde{\pi}_2$. It generates an encoding-decoding token T , where $T := (T_1, T_2)$, each T_i contains a secret parameter $\tilde{\pi}_i$ and its public witness g_i . Given a value and its witness anyone can check if they match. It also generates a set of (additional) public parameters, pp , one of which is e that is a threshold parameter. It generates an empty list, \mathbf{l} . It outputs T, pp , and \mathbf{l} . \mathcal{B} sends $(\tilde{\pi}_1, \tilde{\pi}_2)$ to \mathcal{C} and sends $(g_1, g_2, pp, \mathbf{l})$ to \mathcal{S} .
- $\text{customerInit}(1^\lambda, T, pp) \rightarrow a$. It is a deterministic algorithm run by \mathcal{C} . It takes as input security parameter 1^λ , token T , and set pp of public parameters. It checks the correctness of the elements in T and pp . If the checks pass, it outputs 1. Otherwise, it outputs 0.
- $\text{genUpdateRequest}(T, f, \mathbf{l}) \rightarrow \hat{m}_1^{(C)}$. It is a deterministic algorithm run by \mathcal{C} . It takes as input token T , new payee's detail f , and empty payees' list \mathbf{l} . It generates $m_1^{(C)}$ which is an update request to the payees' list. It uses $T_1 \in T$ and encoding algorithm $\text{Encode}(T_1, \cdot)$ to encode $m_1^{(C)}$ which results in $\hat{m}_1^{(C)}$. It outputs $\hat{m}_1^{(C)}$. Party \mathcal{C} sends the output to \mathcal{S} .
- $\text{insertNewPayee}(\hat{m}_1^{(C)}, \mathbf{l}) \rightarrow \hat{\mathbf{l}}$. It is a deterministic algorithm run by \mathcal{S} . It takes as input \mathcal{C} 's encoded update request $\hat{m}_1^{(C)}$, and \mathcal{C} 's payees' list \mathbf{l} . It inserts the new payee's detail into \mathbf{l} and outputs an updated list, $\hat{\mathbf{l}}$.
- $\text{genWarning}(T, \hat{\mathbf{l}}, aux) \rightarrow \hat{m}_1^{(B)}$. It is run by \mathcal{B} . It takes as input token T , \mathcal{C} 's encoded payees' list $\hat{\mathbf{l}}$, and auxiliary information: aux , e.g., a set of policies. Using $T_1 \in T$, it decodes and checks all elements of the list, e.g., whether they comply with the policies. If the check passes, it sets $m_1^{(B)} = \text{"pass"}$; otherwise, it sets $m_1^{(B)} = \text{warning}$, where the warning is a string containing a warning detail along with the string "warning". It uses T_1 and $\text{Encode}(T_1, \cdot)$ to encode $m_1^{(B)}$ which yields $\hat{m}_1^{(B)}$. It outputs $\hat{m}_1^{(B)}$. Party \mathcal{B} sends $\hat{m}_1^{(B)}$ to \mathcal{S} .
- $\text{genPaymentRequest}(T, in_f, \hat{\mathbf{l}}, \hat{m}_1^{(B)}) \rightarrow \hat{m}_2^{(C)}$. It is run by \mathcal{C} . It takes as input token T , a payment detail in_f , encoded payees' list $\hat{\mathbf{l}}$, and encoded warning message, $\hat{m}_1^{(B)}$. Using $T_1 \in T$, it decodes $\hat{\mathbf{l}}$ and $\hat{m}_1^{(B)}$ yielding \mathbf{l} and $m_1^{(B)}$ respectively. It checks the warning. It sets $m_2^{(C)} = \phi$, if it does not want to proceed. Otherwise, it sets $m_2^{(C)}$ according to the content of in_f and \mathbf{l} (e.g., the amount of payment and payee's detail). It uses T_1 and $\text{Encode}(T_1, \cdot)$ to encode $m_2^{(C)}$ resulting in $\hat{m}_2^{(C)}$. It outputs $\hat{m}_2^{(C)}$. Party \mathcal{C} sends $\hat{m}_2^{(C)}$ to \mathcal{S} .
- $\text{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(B)}$. It is a deterministic algorithm run by \mathcal{B} . It takes as input token T , and encoded payment detail $\hat{m}_2^{(C)}$. Using $T_1 \in T$, it decodes $\hat{m}_2^{(C)}$ and checks the result's validity, e.g., ensures it is well-formed or \mathcal{C} has enough credit. If the check passes, it makes the payment and sets $m_2^{(B)} = \text{"paid"}$. Otherwise, it sets $m_2^{(B)} = \phi$. It uses T_1 and $\text{Encode}(T_1, \cdot)$ to encode $m_2^{(B)}$ yielding $\hat{m}_2^{(B)}$. It outputs $\hat{m}_2^{(B)}$. Party \mathcal{B} sends $\hat{m}_2^{(B)}$ to \mathcal{S} .
- $\text{genComplaint}(\hat{m}_1^{(B)}, \hat{m}_2^{(B)}, T, pk, aux_f) \rightarrow (\hat{z}, \hat{\pi})$. It is run by \mathcal{C} . It takes as input the encoded warning message $\hat{m}_1^{(B)}$, encoded payment message $\hat{m}_2^{(B)}$, token T , public key pk , and auxiliary information aux_f . It initially sets fresh strings (z_1, z_2, z_3) to null. Using $T_1 \in T$, it decodes $\hat{m}_1^{(B)}$ and $\hat{m}_2^{(B)}$ and checks the results' content. If it wants to complain that (i) "pass" message should have been a warning or (ii) no message was provided,

it sets z_1 to “challenge message”. If its complaint is about the warning’s effectiveness, it sets z_2 to a combination of an evidence $u \in \text{aux}_f$, the evidence’s certificate $\text{sig} \in \text{aux}_f$, the certificate’s public parameter, and “challenge warning”, where the certificate is obtained from \mathcal{G} via a query, Q . In certain cases, the certificate might be empty. If its complaint is about the payment, it sets z_3 to “challenge payment”. It uses T_1 and $\text{Encode}(T_1, \cdot)$ to encode $z := (z_1, z_2, z_3)$ and uses $\text{pk}_{\mathcal{D}}$ and another encoding algorithm $\text{Encode}(\text{pk}_{\mathcal{D}}, \cdot)$ to encode $\hat{\pi} := (\hat{\pi}_1, \hat{\pi}_2) \in T$. This results in \hat{z} and $\hat{\pi}$ respectively. It outputs $(\hat{z}, \hat{\pi})$. Party \mathcal{C} sends the pair to \mathcal{S} .

- **verComplaint** $(\hat{z}, \hat{\pi}, g, \hat{\mathbf{m}}, \hat{\mathbf{l}}, j, \text{sk}_{\mathcal{D}}, \text{aux}, pp) \rightarrow \hat{\mathbf{w}}_j$. It is run by every auditor \mathcal{D}_j . It takes as input \mathcal{C} ’s encoded complaint \hat{z} , encoded private parameters $\hat{\pi}$, the tokens’ public parameters $g := (g_1, g_2)$, encoded messages $\hat{\mathbf{m}} = [\hat{m}_1^{(c)}, \hat{m}_2^{(c)}, \hat{m}_1^{(b)}, \hat{m}_2^{(b)}]$, encoded payees’ list $\hat{\mathbf{l}}$, the auditor’s index j , secret key $\text{sk}_{\mathcal{D}}$, auxiliary information aux , and set of public parameters pp . It uses $\text{sk}_{\mathcal{D}}$ to decode $\hat{\pi}$ that yields $\pi := (\pi_1, \pi_2)$. It uses π_1 to decode $\hat{z}, \hat{\mathbf{m}}$, and $\hat{\mathbf{l}}$ that results in $z := (z_1, z_2, z_3), \mathbf{m}$, and \mathbf{l} respectively. It checks if π_i matches g_i . If the check fails, it aborts. It checks if $m_1^{(c)}$ and $m_2^{(c)}$ are non-empty; it aborts if the checks fail. It sets fresh parameters $(w_{1,j}, w_{2,j}, w_{3,j}, w_{4,j})$ to ϕ . If “challenge message” $\in z_1$, given \mathbf{l} , it checks whether “pass” message (in $m_1^{(b)}$) was given correctly or the missing message did not play any role in preventing the fraud. If either checks passes, it sets $w_{1,j} = 0$; otherwise, it sets $w_{1,j} = 1$. If “challenge warning” $\in z_2$, it verifies the certificate in z_2 . If it is invalid, it sets $w_{3,j} = 0$. If it is valid, it sets $w_{3,j} = 1$. It determines the effectiveness of the warning (in $m_1^{(b)}$), by running an algorithm which determines that, i.e., $\text{checkWarning}(\cdot) \in \text{aux}$. If it is effective, i.e., $\text{checkWarning}(m_1^{(b)}) = 1$, it sets its verdict to 0, i.e., $w_{2,j} = 0$; otherwise, it sets $w_{2,j} = 1$. If “challenge payment” $\in z_3$, it checks if the payment was made (with the help of $m_2^{(b)}$). If the check passes, it sets $w_{4,j} = 1$; otherwise, it sets $w_{4,j} = 0$. It uses π_2 to encode $\mathbf{w}_j = [w_{1,j}, w_{2,j}, w_{3,j}, w_{4,j}]$ yielding $\hat{\mathbf{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]$. It outputs $\hat{\mathbf{w}}_j$. Party \mathcal{D}_j sends $\hat{\mathbf{w}}_j$ to \mathcal{S} .
- **resDispute** $(T_2, \hat{\mathbf{w}}, pp) \rightarrow \mathbf{v}$. It is a deterministic algorithm run by \mathcal{DR} . It takes as input token T_2 , auditors’ encoded verdicts $\hat{\mathbf{w}} = [\hat{\mathbf{w}}_1, \dots, \hat{\mathbf{w}}_n]$, and public parameters set pp . It checks if the token’s parameters match. If the check fails, it aborts. It uses $\pi_2 \in T_2$ to decode $\hat{\mathbf{w}}$ and from the result it extracts final verdicts $\mathbf{v} = [v_1, \dots, v_4]$. The extraction procedure ensures each v_i is set to 1 only if at least e auditors’ original verdicts (i.e., $w_{i,j}$) is 1, where $e \in pp$. It outputs \mathbf{v} . If $v_4 = 1$ and (i) either $v_1 = 1$ (ii) or $v_2 = 1$ and $v_3 = 1$, then \mathcal{C} is reimbursed.

Informally, pwdr has two properties; namely, *correctness* and *security*. Correctness requires that, in the absence of a fraudster, the payment journey is completed without the need for the honest customer to complain and the honest bank to reimburse the customer. Below, we formally state it.

Definition 3 (Correctness). A pwdr scheme is correct if the key generation algorithm produces keys $\text{keyGen}(1^\lambda) \rightarrow (\text{sk}, \text{pk})$ such that for any payee’s detail f , payment’s detail in_f , and auxiliary information $(\text{aux}, \text{aux}_f)$, if $\text{bankInit}(1^\lambda) \rightarrow (T, pp, \mathbf{l})$, $\text{customerInit}(1^\lambda, T, pp) \rightarrow a$, $\text{genUpdateRequest}(T, f, \mathbf{l}) \rightarrow \hat{m}_1^{(c)}$, $\text{insertNewPayee}(\hat{m}_1^{(c)}, \mathbf{l}) \rightarrow \hat{\mathbf{l}}$, $\text{genWarning}(T, \hat{\mathbf{l}}, \text{aux}) \rightarrow \hat{m}_1^{(b)}$, $\text{genPaymentRequest}(T, \text{in}_f, \hat{\mathbf{l}}, \hat{m}_1^{(b)}) \rightarrow \hat{m}_2^{(c)}$, $\text{makePayment}(T, \hat{m}_2^{(c)}) \rightarrow \hat{m}_2^{(b)}$, $\text{genComplaint}(\hat{m}_1^{(b)}, \hat{m}_2^{(b)}, T, \text{pk}, \text{aux}_f) \rightarrow (\hat{z}, \hat{\pi}), \forall j \in [n] :$
 $(\text{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{\mathbf{m}}, \hat{\mathbf{l}}, j, \text{sk}_{\mathcal{D}}, \text{aux}, pp) \rightarrow \hat{\mathbf{w}}_j), \text{resDispute}(T_2, \hat{\mathbf{w}}, pp) \rightarrow \mathbf{v}$, then $(z_1 = z_2 = z_3 = \phi) \wedge (\mathbf{v} = 0)$, where $g := (g_1, g_2) \in T$, $\hat{\mathbf{m}} = [\hat{m}_1^{(c)}, \hat{m}_2^{(c)}, \hat{m}_1^{(b)}, \hat{m}_2^{(b)}]$, $\hat{\mathbf{w}} = [\hat{\mathbf{w}}_1, \dots, \hat{\mathbf{w}}_n]$, and $z := (z_1, z_2, z_3)$ is the result of decoding \hat{z} .

A pwdr scheme is secure if it satisfies three main properties; namely, (a) security against a malicious victim, (b) security against a malicious bank, and (c) privacy. Below, we formally define them. Intuitively, security against a malicious victim requires that the victim of an APP fraud that is not qualified for the reimbursement should not be reimbursed (despite it tries to be). More specifically, a corrupt victim cannot (a) make at least the threshold of the committee members, \mathcal{D}_j s, conclude that \mathcal{B} should have provided a warning, although \mathcal{B} has done so, or (b) make \mathcal{DR} conclude that the pass message was incorrectly given or a vital warning message was missing despite only less than the threshold of \mathcal{D}_j s believing so, or (c) persuade at least the threshold of \mathcal{D}_j s to conclude that the warning was ineffective although it was effective, or (d) make \mathcal{DR} believe that the warning message was ineffective although only less than the threshold of \mathcal{D}_j s believe that, or (e) convince \mathcal{D}_j s to accept an invalid certificate, or (f) make \mathcal{DR} believe that at least the threshold

of \mathcal{D}_j s accepted the certificate although they did not, except for a negligible probability. Below, we formally state it.

Definition 4 (Security against a malicious victim). A pwr scheme is secure against a malicious victim, if for any security parameter λ , auxiliary information aux , and probabilistic polynomial-time adversary \mathcal{A} , there exists a negligible function $\mu(\cdot)$, such that for an experiment $\text{Exp}_1^{\mathcal{A}}$:

$$\begin{array}{l} \text{Exp}_1^{\mathcal{A}}(1^\lambda, aux) \\ \text{keyGen}(1^\lambda) \rightarrow (sk, pk) \\ \text{bankInit}(1^\lambda) \rightarrow (T, pp, \mathbf{l}) \\ \mathcal{A}(1^\lambda, T, pp, \mathbf{l}) \rightarrow \hat{m}_1^{(c)} \\ \text{insertNewPayee}(\hat{m}_1^{(c)}, \mathbf{l}) \rightarrow \hat{\mathbf{l}} \\ \text{genWarning}(T, \hat{\mathbf{l}}, aux) \rightarrow \hat{m}_1^{(s)} \\ \mathcal{A}(T, \hat{\mathbf{l}}, \hat{m}_1^{(s)}) \rightarrow \hat{m}_2^{(c)} \\ \text{makePayment}(T, \hat{m}_2^{(c)}) \rightarrow \hat{m}_2^{(s)} \\ \mathcal{A}(\hat{m}_1^{(s)}, \hat{m}_2^{(s)}, T, pk) \rightarrow (\hat{z}, \hat{\pi}) \\ \forall j, j \in [n] : \\ \left(\text{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{\mathbf{m}}, \hat{\mathbf{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\mathbf{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}] \right) \\ \text{resDispute}(T_2, \hat{\mathbf{w}}, pp) \rightarrow \mathbf{v} = [v_1, \dots, v_4] \end{array}$$

it holds that:

$$\Pr \left[\begin{array}{l} \left((m_1^{(s)} = \text{warning}) \wedge \left(\sum_{j=1}^n w_{1,j} \geq e \right) \right) \\ \vee \left(\left(\sum_{j=1}^n w_{1,j} < e \right) \wedge (v_1 = 1) \right) \\ \vee \left((\text{checkWarning}(m_1^{(s)}) = 1) \wedge \left(\sum_{j=1}^n w_{2,j} \geq e \right) \right) \\ \vee \left(\left(\sum_{j=1}^n w_{2,j} < e \right) \wedge (v_2 = 1) \right) \\ \vee \left(u \notin Q \wedge \text{Sig.ver}(pk, u, sig) = 1 \right) \\ \vee \left(\left(\sum_{j=1}^n w_{3,j} < e \right) \wedge (v_3 = 1) \right) \end{array} : \text{Exp}_1^{\mathcal{A}}(\text{input}) \right] \leq \mu(\lambda),$$

where $g := (g_1, g_2) \in T$, $\hat{\mathbf{m}} = [\hat{m}_1^{(c)}, \hat{m}_2^{(c)}, \hat{m}_1^{(s)}, \hat{m}_2^{(s)}]$, $(w_{1,j}, \dots, w_{3,j})$ are the result of decoding $(\hat{w}_{1,j}, \dots, \hat{w}_{3,j}) \in \hat{\mathbf{w}}$, $\text{checkWarning}(\cdot)$ determines a warning's effectiveness, $\text{input} := (1^\lambda, aux)$, $sk_{\mathcal{D}} \in sk$, and n is the total number of auditors. The probability is taken over the uniform choice of sk , randomness used in the blockchain's primitives (e.g., in signatures), randomness used in the encoding, and the randomness of \mathcal{A} .

Note, in the above experiment, we did not explicitly pass payment detail f and auxiliary information (in_f, aux_f) on to the adversary, to let the adversary pick them (on the client's behalf).

Intuitively, security against a malicious bank requires that a malicious bank should not be able to disqualify an honest victim of an APP fraud from being reimbursed. Specifically, a corrupt bank cannot (a) make \mathcal{DR} conclude that the “pass” message was correctly given or an important warning message was not missing despite at least the threshold of \mathcal{D}_j s do not believe so, or (b) convince \mathcal{DR} that the warning message was effective although at least the threshold of \mathcal{D}_j s do not believe so, or (c) make \mathcal{DR} believe that less than the threshold of \mathcal{D}_j s did not accept the certificate although at least the threshold of them did that, or (d) make \mathcal{DR} believe that no payment was made, although at least the threshold of \mathcal{D}_j s believe the opposite, except for a negligible probability. Below, we formally state it.

Definition 5 (Security against a malicious bank). A pwdr scheme is secure against a malicious bank, if for any security parameter λ , auxiliary information aux , and probabilistic polynomial-time adversary \mathcal{A} , there exists a negligible function $\mu(\cdot)$, such that for an experiment $\text{Exp}_2^{\mathcal{A}}$:

$\text{Exp}_2^{\mathcal{A}}(1^\lambda, aux)$

```

keyGen( $1^\lambda$ )  $\rightarrow$  ( $sk, pk$ )
 $\mathcal{A}(1^\lambda) \rightarrow (T, pp, \mathbf{l}, f, in_f, aux_f)$ 
customerInit( $1^\lambda, T, pp$ )  $\rightarrow a$ 
genUpdateRequest( $T, f, \mathbf{l}$ )  $\rightarrow \hat{m}_1^{(c)}$ 
insertNewPayee( $\hat{m}_1^{(c)}, \mathbf{l}$ )  $\rightarrow \hat{\mathbf{l}}$ 
 $\mathcal{A}(T, \hat{\mathbf{l}}, aux) \rightarrow \hat{m}_1^{(B)}$ 
genPaymentRequest( $T, in_f, \hat{\mathbf{l}}, \hat{m}_1^{(B)}$ )  $\rightarrow \hat{m}_2^{(c)}$ 
 $\mathcal{A}(T, \hat{m}_2^{(c)}) \rightarrow \hat{m}_2^{(B)}$ 
genComplaint( $\hat{m}_1^{(B)}, \hat{m}_2^{(B)}, T, pk, aux_f$ )  $\rightarrow (\hat{z}, \hat{\pi})$ 
 $\forall j, j \in [n] :$ 
  ( $\text{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{\mathbf{m}}, \hat{\mathbf{l}}, j, sk_D, aux, pp) \rightarrow \hat{\mathbf{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]$ )
resDispute( $T_2, \hat{\mathbf{w}}, pp$ )  $\rightarrow \mathbf{v} = [v_1, \dots, v_4]$ 

```

it holds that:

$$\Pr \left[\begin{array}{l} \left(\sum_{j=1}^n w_{1,j} \geq e \right) \wedge (v_1 = 0) \\ \vee \left(\sum_{j=1}^n w_{2,j} \geq e \right) \wedge (v_2 = 0) \\ \vee \left(\sum_{j=1}^n w_{3,j} \geq e \right) \wedge (v_3 = 0) \\ \vee \left(\sum_{j=1}^n w_{4,j} \geq e \right) \wedge (v_4 = 0) \end{array} : \text{Exp}_2^{\mathcal{A}}(\text{input}) \right] \leq \mu(\lambda),$$

where $g := (g_1, g_2) \in T$, $\hat{\mathbf{m}} = [\hat{m}_1^{(c)}, \hat{m}_2^{(c)}, \hat{m}_1^{(B)}, \hat{m}_2^{(B)}]$, $(w_{1,j}, \dots, w_{3,j})$ are the result of decoding $(\hat{w}_{1,j}, \dots, \hat{w}_{3,j}) \in \hat{\mathbf{w}}$, $\text{input} := (1^\lambda, aux)$, $sk_D \in sk$, n is the total number of auditors. The probability is taken over the uniform choice of sk , randomness used in the blockchain's primitives, randomness used during the encoding, and the randomness of \mathcal{A} .

Now we move on to privacy. Informally, a pwdr is privacy-preserving if it protects the privacy of (1) the customers', bank's, and auditors' messages (except public parameters) from non-participants of the scheme, including other customers, and (2) each auditor's verdict from \mathcal{DR} which sees the final verdict. Below, we formally state it.

Definition 6 (Privacy). A pwdr preserves privacy if the following two properties are satisfied.

1. For any probabilistic polynomial-time adversary \mathcal{A}_1 , security parameter λ , and auxiliary information aux , there exists a negligible function $\mu(\cdot)$, such that for any experiment $\text{Exp}_3^{\mathcal{A}_1}$:

$\text{Exp}_3^{A_1}(1^\lambda, aux)$

```

keyGen( $1^\lambda$ )  $\rightarrow$  ( $sk, pk$ )
bankInit( $1^\lambda$ )  $\rightarrow$  ( $T, pp, l$ )
customerInit( $1^\lambda, T, pp$ )  $\rightarrow a$ 
 $\mathcal{A}_1(1^\lambda, pk, a, pp, g, l) \rightarrow ((f_0, f_1), (in_{f_0}, in_{f_1}), (aux_{f_0}, aux_{f_1}))$ 
 $\gamma \xleftarrow{\$} \{0, 1\}$ 
genUpdateRequest( $T, f_\gamma, l$ )  $\rightarrow \hat{m}_1^{(c)}$ 
insertNewPayee( $\hat{m}_1^{(c)}, l$ )  $\rightarrow \hat{l}$ 
genWarning( $T, \hat{l}, aux$ )  $\rightarrow \hat{m}_1^{(B)}$ 
genPaymentRequest( $T, in_{f_\gamma}, \hat{l}, \hat{m}_1^{(B)}$ )  $\rightarrow \hat{m}_2^{(c)}$ 
makePayment( $T, \hat{m}_2^{(c)}$ )  $\rightarrow \hat{m}_2^{(B)}$ 
genComplaint( $\hat{m}_1^{(B)}, \hat{m}_2^{(B)}, T, pk, aux_{f_\gamma}$ )  $\rightarrow (\hat{z}, \hat{\pi})$ 
 $\forall j, j \in [n] :$ 
  ( $\text{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{m}, \hat{l}, j, sk_D, aux, pp) \rightarrow \hat{w}_j$ )
resDispute( $T_2, \hat{w}, pp$ )  $\rightarrow v$ 

```

it holds that:

$$\Pr [\mathcal{A}_1(g, \hat{m}, \hat{l}, \hat{z}, \hat{\pi}, \hat{w}) \rightarrow \gamma : \text{Exp}_3^{A_1}(\text{input})] \leq \frac{1}{2} + \mu(\lambda).$$

2. For any probabilistic polynomial-time adversaries \mathcal{A}_2 and \mathcal{A}_3 , security parameter λ , and auxiliary information aux , there exists a negligible function $\mu(\cdot)$, such that for any experiment $\text{Exp}_4^{A_2}$:

$\text{Exp}_4^{A_2}(1^\lambda, aux)$

```

keyGen( $1^\lambda$ )  $\rightarrow$  ( $sk, pk$ )
bankInit( $1^\lambda$ )  $\rightarrow$  ( $T, pp, l$ )
customerInit( $1^\lambda, T, pp$ )  $\rightarrow a$ 
 $\mathcal{A}_2(1^\lambda, pk, a, pp, l) \rightarrow (f, in_f, aux_f)$ 
genUpdateRequest( $T, f, l$ )  $\rightarrow \hat{m}_1^{(c)}$ 
insertNewPayee( $\hat{m}_1^{(c)}, l$ )  $\rightarrow \hat{l}$ 
 $\mathcal{A}_2(T, \hat{l}, aux) \rightarrow m_1^{(B)}$ 
Encode( $T_1, m_1^{(B)}$ )  $\rightarrow \hat{m}_1^{(B)}$ 
genPaymentRequest( $T, in_f, \hat{l}, \hat{m}_1^{(B)}$ )  $\rightarrow \hat{m}_2^{(c)}$ 
 $\mathcal{A}_2(T, pk, aux_f, \hat{m}_1^{(B)}, \hat{m}_2^{(c)}) \rightarrow (m_2^{(B)}, z, \hat{\pi})$ 
Encode( $T_1, m_2^{(B)}$ )  $\rightarrow \hat{m}_2^{(B)}$ 
Encode( $T_1, z$ )  $\rightarrow \hat{z}$ 
Encode( $pk_D, \hat{\pi}$ )  $\rightarrow \hat{\pi}$ 
 $\forall j, j \in [n] :$ 
  ( $\text{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{m}, \hat{l}, j, sk_D, aux, pp) \rightarrow \hat{w}_j$ )
resDispute( $T_2, \hat{w}, pp$ )  $\rightarrow v$ 

```

it holds that:

$$\Pr [\mathcal{A}_3(T_2, pk, pp, g, \hat{m}, \hat{l}, \hat{z}, \hat{\pi}, \hat{w}, v) \rightarrow w_j : \text{Exp}_4^{A_2}(\text{input})] \leq Pr' + \mu(\lambda),$$

where $g := (g_1, g_2) \in T$, $\hat{m} = [\hat{m}_1^{(c)}, \hat{m}_2^{(c)}, \hat{m}_1^{(B)}, \hat{m}_2^{(B)}]$, $\hat{w} = [\hat{w}_1, \dots, \hat{w}_n]$, $\text{input} := (1^\lambda, aux)$, $T_1 \in T$, $pk_D \in pk$, $sk_D \in sk$, n is the total number of auditors. Moreover, Pr' is defined as follows. Let auditor \mathcal{D}_i output

0 and 1 with probabilities $Pr_{i,0}$ and $Pr_{i,1}$ respectively. Then, Pr' is defined as $Max(Pr_{1,0}, Pr_{1,1}, \dots, Pr_{n,0}, Pr_{n,1})$. In the above privacy definition, the probability is taken over the uniform choice of sk , the probability that each \mathcal{D}_j outputs 0 or 1, the randomness used in the blockchain's primitives, the randomness used during the encoding, and the randomness of \mathcal{A}_1 in $\text{Exp}_3^{\mathcal{A}_1}$ and \mathcal{A}_2 in $\text{Exp}_4^{\mathcal{A}_2}$.

Definition 7 (Security). A pwdr is secure if it meets security against a malicious victim, security against a malicious bank, and preserves privacy with respect to definitions 4, 5, and 6 respectively.

6 Payment with Dispute Resolution Protocol

In this section, first we provide an outline of the PwDR protocol. Then, we present several subroutines that will be used in this protocol. After that, we describe the PwDR protocol in detail.

6.1 An Overview of the PwDR Protocol

In this section, we provide an overview of the PwDR. Initially, \mathcal{C} and \mathcal{B} agree on a smart contract \mathcal{S} . They also use the SAP to agree on two private statements including two secret keys that will be used to encrypt outgoing messages. When \mathcal{C} wants to transfer money to a new payee, it signs into its online banking. It generates an update request (that specifies the new payee's detail), encrypts it, and sends the result to \mathcal{S} . Then, \mathcal{B} decrypts and checks the request, e.g., whether it meets its internal policy. Depending on the request, \mathcal{B} generates a pass or warning message. It encrypts the message and sends the result to \mathcal{S} . Next, \mathcal{C} checks \mathcal{B} 's message and decides whether to make payment. If it decides to do so, it sends an encrypted payment detail to \mathcal{S} . After that, \mathcal{B} decrypts the message and locally transfers the amount of money specified in \mathcal{C} 's message. Once the money is transferred, \mathcal{B} sends an encrypted "paid" message to \mathcal{S} .

Once \mathcal{C} realises that it has fallen victim, it raises a dispute. Specifically, it generates an encrypted complaint that can challenge the effectiveness of the warning and/or any payment inconsistency. It can include in the complaint an evidence/certificate, e.g., asserting that it falls into the vulnerable customer category as defined in the CRM code. \mathcal{C} encrypts the complaint and sends to \mathcal{S} the result and a proof asserting the secret key's correctness. Then, each auditor verifies the proof. If the verification passes, it decrypts and compiles \mathcal{C} 's complaint to generate a (set of) verdict. Each auditor encodes its verdict and sends the encoded verdict's encryption to \mathcal{S} . To resolve a dispute between \mathcal{C} and \mathcal{B} , either of them invokes \mathcal{DR} . To do so, it directly sends to \mathcal{DR} one of the above secret keys and a proof asserting that key was generated correctly. \mathcal{DR} verifies the proof. If approved, it locally decrypts the encrypted encoded verdicts (retrieved from \mathcal{S}) and finds out the final verdict. If the final verdict indicates the legitimacy of \mathcal{C} 's complaint, then \mathcal{C} must be reimbursed. Note, the verdicts are encoded in a way that even after decrypting them, \mathcal{DR} cannot link a verdict to a committee member or even figure out how many 1 or 0 verdicts were provided (except when all verdicts are 0). However, it can find out whether at least the threshold of the auditors voted in favour of \mathcal{C} .

6.2 A Subroutine for Determining Bank's Message Status

In the payment journey, the customer may receive a "pass" message or even nothing at all, e.g., due to a system failure. In such cases, a victim must be able to complain that if the pass or missing message was a warning, then it would have prevented it from falling victim. To assist the auditors to deal with such complaints deterministically, we propose `verStat(.)` algorithm, which is run locally by each committee member. This algorithm is presented in Figure 1.

6.3 A Subroutine for Checking a Warning's Effectiveness

To help the auditors deterministically compile a victim's complaint about a warning's effectiveness, we propose an algorithm, called `checkWarning(.)` which is run locally by each auditor. It also allows the victims to provide (to the auditors) a certificate/evidence as part of their complaints. This algorithm is presented in Figure 2.

$\text{verStat}(add_{\mathcal{S}}, m^{(\mathcal{B})}, \mathbf{l}, \Delta, aux) \rightarrow w_1$

- *Input.* $add_{\mathcal{S}}$: the address of smart contract \mathcal{S} , $m^{(\mathcal{B})}$: \mathcal{B} 's warning message, \mathbf{l} : customer's payees' list, Δ : a time parameter, and aux : auxiliary information, e.g., bank's policy.
 - *Output.* $w_1 = 0$: if the “pass” message had been given correctly or the missing message did not play any role in preventing the fraud; $w_1 = 1$: otherwise.
1. reads the content of \mathcal{S} . It checks if $m^{(\mathcal{B})} = \text{“pass”}$ or the encrypted warning message was not sent on time (i.e., never sent or sent after $t_0 + \Delta$). If one of the checks passes, it proceeds; Otherwise, it aborts.
 2. checks the validity of customer's most recent payees' list \mathbf{l} , with the help of aux .
 - if \mathbf{l} contains an invalid element, it sets $w_1 = 1$.
 - otherwise, it sets $w_1 = 0$.
 3. returns w_1 .

Fig. 1: Algorithm to Determine a Bank's Message Status

$\text{checkWarning}(add_{\mathcal{S}}, z, m^{(\mathcal{B})}, aux') \rightarrow (w_2, w_3)$

- *Input.* $add_{\mathcal{S}}$: the address of smart contract \mathcal{S} , z : \mathcal{C} 's complaint, $m^{(\mathcal{B})}$: \mathcal{B} 's warning message, and aux' : auxiliary information, e.g., guideline on warnings' effectiveness.
 - *Output.* $w_2 = 0$: if the given warning message is effective; $w_2 = 1$: if the warning message is ineffective. Also, $w_3 = 1$: if the certificate in z is valid or no certificate is provided; $w_3 = 0$: if the certificate is invalid.
1. parse $z = m || sig || pk || \text{“challenge warning”}$. If sig is empty, it sets $w_3 = 0$ and goes to step 2. Otherwise, it:
 - (a) verifies the certificate: $\text{Sig.ver}(pk, m, sig) \rightarrow h$.
 - (b) if the certificate is rejected (i.e., $h = 0$), it sets $w_3 = 0$. It goes to step 4.
 - (c) otherwise (i.e., $h = 1$), it sets $w_3 = 1$ and moves onto the next step.
 2. checks if “warning” $\in m^{(\mathcal{B})}$. If the check is passed, it proceeds to the next step. Otherwise, it aborts.
 3. checks the warning's effectiveness, with the assistance of the evidence m and auxiliary information aux' .
 - if it is effective, it sets $w_2 = 0$.
 - otherwise, it sets $w_2 = 1$.
 4. returns (w_2, w_3) .

Fig. 2: Algorithm to Check Warning's Effectiveness

6.4 Subroutines for Encoding-Decoding Verdicts

In this section, we present verdict encoding and decoding protocols. They let a third party \mathcal{I} , e.g., \mathcal{DR} , learn if a threshold of the auditors voted 1, while satisfying the following requirements. The protocols (1) generate unlinkable verdicts, (2) do not require auditors to interact with each other for each customer, and (3) are efficient. Since the second and third requirements are self-explanatory, we only explain the first one. Informally, the first property states that the protocols generate encoded verdicts and final verdict in a way that \mathcal{I} , given these values, cannot (a) link a verdict to an auditor (except when all verdicts are 0), and (b) learn the total number of 1 or 0 verdicts when they provide different verdicts. Shortly, we present two variants of verdict encoding and decoding protocol. The first variant is highly efficient and suitable when the threshold is 1. The second variant is generic and works for any threshold (but is less efficient).

Variant 1: Efficient Verdict Encoding-Decoding Protocol. This variant has two protocols, Private Verdict Encoding (PVE) and Final Verdict Decoding (FVD). They let \mathcal{I} learn if at least one auditor voted 1. This variant relies on our observation that if a set of random values and 0s are XORed, then the result reveals nothing, e.g., about the number of non-zero and zero values. Below, we formally state it. We refer readers to Appendix A for the proof.

Theorem 1. *Let set $S = \{s_1, \dots, s_m\}$ be the union of two disjoint sets S' and S'' , where S' contains non-zero random values pick uniformly from a finite field \mathbb{F}_p , S'' contains zeros, $|S'| \geq c' = 1$, $|S''| \geq c'' = 0$, and pair (c', c'') is public information. Then, $r = \bigoplus_{i=1}^m s_i$ reveals nothing beyond the public information.*

At a high level, PVE and FVD work as follows. The auditors only once (for all future customers) agree on a secret key. This key will let each of them, in PVE, generate a pseudorandom masking value such that if all masking values are XORed, they would cancel out each other.² In PVE, each auditor encodes its verdict by (i) representing it as a parameter which is set to 0 if the verdict is 0, or to a random value if the verdict is 1, and then (ii) “masking” this parameter with the above pseudorandom value. It sends the result to \mathcal{I} . In FVD, \mathcal{I} XORs all encoded verdicts. This removes the masks and XORs all verdicts’ representations. If the result is 0, it concludes that all auditors voted 0; so, the final verdict is 0. But, if the result is not 0, it knows that at least one of the auditors voted 1, so the final verdict is 1. Figures 3 and 4 present PVE and FVD respectively.

Intuitively, the primary reason this variant meets property 1 is that each masked verdict reveals nothing about the verdict (and its representation) and given the final verdict, \mathcal{I} cannot distinguish between the case where there is exactly one auditor that voted 1 and the case where multiple auditors voted 1, as in both cases \mathcal{I} extracts only a single random value, which reveals nothing about the number of auditors which voted 0 or 1 (due to Theorem 1). It is evident that this variant meets property 2 as the auditors interact with each other *only once* (to agree on a key) for all customers. It also meets property 3 as it involves pseudorandom function invocations and XOR operations which are highly efficient operations. Note, the protocols’ correctness holds with an overwhelming probability, i.e., $1 - \frac{1}{2^\lambda}$. Specifically, if two auditors represent their verdict by an identical random value, then when they are XORed they would cancel out each other which can affect the result’s correctness. The same holds if the XOR of multiple verdicts’ representations results in a value that can cancel out another verdict’s representation. Nevertheless, the probability that such an event occurs is negligible in the security parameter $|p| = \lambda$, i.e., the probability is at most $\frac{1}{2^\lambda}$.

Variant 2: Generic Verdict Encoding-Decoding Protocol. This variant also includes two protocols, Generic Private Verdict Encoding (GPVE) and Generic Final Verdict Decoding (GFVD) which let \mathcal{I} learn if at least e auditors voted 1, where e is an integer in $[1, n]$. It uses a novel combination of Bloom filter and combinatorics. It relies on our observation that a Bloom filter encoding a set of random values reveals nothing about the set’s elements. We formally state it in Theorem 2 whose proof is given in Appendix B.

² This is similar to the idea used in the XOR-based secret sharing [34].

PVE($\bar{k}_0, \text{ID}, w_j, o, n, j$) $\rightarrow \bar{w}_j$

- *Input.* \bar{k}_0 : a key of pseudorandom function $\text{PRF}(\cdot)$, ID : a unique identifier, w_j : a verdict, o : an offset, n : the total number of auditors, and j : an auditor's index.
- *Output.* \bar{w}_j : an encoded verdict.

Auditor \mathcal{D}_j takes the following steps.

1. computes a pseudorandom value, as follows.
 - if $j < n$: $r_j = \text{PRF}(\bar{k}_0, o || j || \text{ID})$.
 - if $j = n$: $r_j = \bigoplus_{i=1}^{n-1} r_i$.
2. sets a fresh parameter, w'_j , as below.

$$w'_j = \begin{cases} 0, & \text{if } w_j = 0 \\ \alpha_j \xleftarrow{\$} \mathbb{F}_p, & \text{if } w_j = 1 \end{cases}$$

3. encodes w'_j as follows. $\bar{w}_j = w'_j \oplus r_j$.
4. outputs \bar{w}_j .

Fig. 3: Private Verdict Encoding (PVE) Protocol

FVD($n, \bar{\mathbf{w}}$) $\rightarrow v$

- *Input.* n : the total number of auditors, and $\bar{\mathbf{w}} = [\bar{w}_1, \dots, \bar{w}_n]$: a vector of all auditors' encoded verdicts.
- *Output.* v : final verdict.

A third-party \mathcal{I} takes the following steps.

1. combines all auditors' encoded verdicts, $\bar{w}_j \in \bar{\mathbf{w}}$, as follows. $c = \bigoplus_{j=1}^n \bar{w}_j$
2. sets the final verdict v depending on the content of c . Specifically,

$$v = \begin{cases} 0, & \text{if } c = 0 \\ 1, & \text{otherwise} \end{cases}$$

3. outputs v .

Fig. 4: Final Verdict Decoding (FVD) Protocol

Theorem 2. *Let $S = \{s_1, \dots, s_m\}$ be a set of random values picked uniformly from finite field \mathbb{F}_p , where the cardinality of S is public information. Let BF be a Bloom filter encoding all elements of S . Then, BF reveals nothing about any element of S , beyond the public information, except with a negligible probability in the security parameter λ , i.e., with a probability at most $\frac{|S|}{2^\lambda}$.*

Now we present an overview of this variant. In this variant also, the auditors initially agree on a secret key used to generate a pseudorandom masking value. Each auditor \mathcal{D}_j represents its verdict by a parameter,

such that if its verdict is 0, it sets the parameter to 0; but, if the verdict is 1, it sets the parameter to a fresh *pseudorandom* value α_j (instead of a random value used in Variant 1) where this pseudorandom value is also derived from the above key. Thus, there would be a set $A = \{\alpha_1, \dots, \alpha_n\}$ from which \mathcal{D}_j would pick α_j to represent its verdict 1. Each \mathcal{D}_j masks its verdict representation by its masking value. It sends the result to \mathcal{I} . Also, (only) auditor \mathcal{D}_n generates a set W of all combinations of auditors' verdict 1's representations that satisfy the threshold, e . Specifically, for every integer i in $[e, n]$, it computes the combinations (without repetition) of i elements from $A = \{\alpha_1, \dots, \alpha_n\}$. If multiple elements are taken at a time (i.e., $i > 1$), they are XORed with each other. Note, \mathcal{D}_n computes these values regardless of what a specific auditor votes. Let $W = \{(\alpha_1 \oplus \dots \oplus \alpha_e), (\alpha_2 \oplus \dots \oplus \alpha_{e+1}), \dots, (\alpha_1 \oplus \dots \oplus \alpha_n)\}$ be the result. For instance, if the total number of arbiters, n , is 3 and the threshold, e , is 2, then $W = \{(\alpha_1 \oplus \alpha_2), (\alpha_2 \oplus \alpha_3), (\alpha_1 \oplus \alpha_3), (\alpha_1 \oplus \alpha_2 \oplus \alpha_n)\}$. After that, it generates an empty Bloom filter and inserts all elements of W into this Bloom filter. Let \mathbf{BF} be the Bloom filter that encodes W 's elements. It sends \mathbf{BF} to \mathcal{I} . Note that inserting the combinations into \mathbf{BF} ensures that the privacy of each vote's representation (e.g., α_j) is protected from \mathcal{I} .

In GFVD, to decode and extract the final verdict, \mathcal{I} XORs all masked verdict representations which removes the masking values and XORs the representations. Let c be the result. If $c = 0$, then \mathcal{I} concludes that all auditors voted 0; so, it sets the final verdict to 0. If $c \neq 0$, then it checks if $c \in \mathbf{BF}$. If it is, then it concludes that at least the threshold of the auditors voted 1, so it sets the final verdict to 1. Otherwise ($c \notin \mathbf{BF}$), it learns that less than the threshold of the auditors voted 1; so, it sets the final verdict to 0. Figures 5 and 6 present the GPVE and GFVD protocols in detail.

GPVE($\bar{k}_0, \text{ID}, w_j, o, e, n, j$) \rightarrow (\bar{w}_j, \mathbf{BF})

- *Input.* \bar{k}_0 : a key of pseudorandom function $\text{PRF}(\cdot)$, ID : a unique identifier, w_j : a verdict, o : an offset, e : a threshold, n : the total number of auditors, and j : an auditor's index.
- *Output.* \bar{w}_j : an encoded verdict.

Auditor \mathcal{D}_j takes the following steps.

1. computes a pseudorandom value, as follows.
 - if $j < n$: $r_j = \text{PRF}(\bar{k}_0, 1||o||j||\text{ID})$.

- if $j = n$: $r_j = \bigoplus_{i=1}^{n-1} r_i$.

Note, the above second step is taken only by \mathcal{D}_n .

2. sets a fresh parameter, w'_j , that represents a verdict, as below.

$$w'_j = \begin{cases} 0, & \text{if } w_j = 0 \\ \alpha_j = \text{PRF}(\bar{k}_0, 2||o||j||\text{ID}), & \text{if } w_j = 1 \end{cases}$$

3. masks w'_j as follows. $\bar{w}_j = w'_j \oplus r_j$.
4. if $j = n$, computes a Bloom filter that encodes the combinations of verdict representations (i.e., w'_j) for verdict 1. In particular, it takes the following steps.
 - for every integer i in the range $[e, n]$, computes the combinations (without repetition) of i elements from set $\{\alpha_1, \dots, \alpha_n\}$. In the case where multiple elements are taken at a time (i.e., $i > 1$), the elements are XORed with each other. Let $W = \{(\alpha_1 \oplus \dots \oplus \alpha_e), (\alpha_2 \oplus \dots \oplus \alpha_{e+1}), \dots, (\alpha_1 \oplus \dots \oplus \alpha_n)\}$ be the result.
 - constructs an empty Bloom filter. Then, it inserts all elements of W into this Bloom filter. Let \mathbf{BF} be the Bloom filter encoding W 's elements.
5. outputs (\bar{w}_j, \mathbf{BF}).

Fig. 5: Generic Private Verdict Encoding (GPVE) Protocol

GFVD(n, \bar{w}, BF) $\rightarrow v$

- *Input.* n : the total number of auditors, and $\bar{w} = [\bar{w}_1, \dots, \bar{w}_n]$: a vector of all auditors' encoded verdicts.
- *Output.* v : final verdict.

A third-party \mathcal{I} takes the following steps.

1. combines all auditors' encoded verdicts, $\bar{w}_j \in \bar{w}$, as follows. $c = \bigoplus_{j=1}^n \bar{w}_j$
2. checks if c is in the Bloom filter, BF .
3. sets the final verdict v depending on the content of c . Specifically,

$$v = \begin{cases} 0, & \text{if } c = 0 \text{ or } c \notin \text{BF} \\ 1, & \text{if } c \in \text{BF} \end{cases}$$

4. outputs v .

Fig. 6: Generic Final Verdict Decoding (GFVD) Protocol

Note also that the total number of the combinations (i.e., the cardinality of W) is relatively small when the number of arbiters is not very high. In general, due to the binomial theorem, the cardinality of W is determined as follows:

$$|W| = \sum_{i=e}^n \frac{n!}{i!(n-i)!} \quad (1)$$

For instance, when $n = 10$ and $e = 6$, then W 's cardinality is only 386, according to Equation 1. In the above scheme, instead of inserting the combinations into BF we could simply hash the combinations and give the hash values to \mathcal{I} . However, using a Bloom filter allows us to save considerable communication costs. For instance, when $n = 10$, $e = 6$, and SHA-256 is used, then \mathcal{D}_n needs to send $98816 = 386 \times 256$ bits to \mathcal{I} , whereas if they are inserted into a Bloom filter, then it only needs to send 22276 bits to \mathcal{I} . Thus, by using a Bloom filter, it can save communication costs by at least a factor of 4. We refer readers to Appendix C for further discussion on the verdict encoding-decoding protocols.

6.5 The PwDR Protocol

In this section, we present the PwDR protocol in detail.

1. *Generating \mathcal{G} 's and \mathcal{D}_j 's Parameters:* $\text{keyGen}(1^\lambda) \rightarrow (sk, pk)$.

Parties \mathcal{G} and \mathcal{D}_j take steps 1a and 1b respectively.

- (a) calls $\text{Sig.keyGen}(1^\lambda) \rightarrow (sk_{\mathcal{G}}, pk_{\mathcal{G}})$ to generate secret key $sk_{\mathcal{G}}$ and public key $pk_{\mathcal{G}}$. It publishes $pk_{\mathcal{G}}$.
- (b) calls $\text{keyGen}(1^\lambda) \rightarrow (sk_{\mathcal{D}}, pk_{\mathcal{D}})$ to generate decrypting secret key $sk_{\mathcal{D}}$ and encrypting public key $pk_{\mathcal{D}}$. It publishes the public key $pk_{\mathcal{D}}$ and sends $sk_{\mathcal{D}}$ to the rest of the auditors.

Let $sk := (sk_{\mathcal{G}}, sk_{\mathcal{D}})$ and $pk := (pk_{\mathcal{G}}, pk_{\mathcal{D}})$. Note, this phase takes place only once for all customers.

2. *Bank-side Initiation:* $\text{bankInit}(1^\lambda) \rightarrow (T, pp, \mathbf{l})$.

Bank \mathcal{B} takes the following steps.

- (a) picks secret keys \bar{k}_1 and \bar{k}_2 for the symmetric key encryption scheme and pseudorandom function PRF respectively. It sets two private statements as $\pi_1 = \bar{k}_1$ and $\pi_2 = \bar{k}_2$.
- (b) calls $\text{SAP.init}(1^\lambda, \text{adr}_{\mathcal{B}}, \text{adr}_{\mathcal{C}}, \pi_i) \rightarrow (r_i, g_i, \text{adr}_{\text{SAP}})$ to initiate agreements on statements $\pi_i \in \{\pi_1, \pi_2\}$ with \mathcal{C} . Let $T_i := (\bar{\pi}_i, g_i)$ and $T := (T_1, T_2)$, where $\bar{\pi}_i := (\pi_i, r_i)$ is the opening of g_i . It also sets parameter Δ as a time window between two specific time points, i.e., $\Delta = t_i - t_{i-1}$. Briefly, it is used to impose an upper bound on a message delay.

- (c) sends $\ddot{\pi} := (\ddot{\pi}_1, \ddot{\pi}_2)$ to \mathcal{C} and sends public parameter $pp := (adr_{\text{SAP}}, \Delta)$ to smart contract \mathcal{S} .
3. Customer-side Initiation: $\text{customerInit}(1^\lambda, T, pp) \rightarrow a$.
Customer \mathcal{C} takes the following steps.
- calls $\text{SAP.agree}(\pi_i, r_i, g_i, adr_{\mathcal{B}}, adr_{\text{SAP}}) \rightarrow (g'_i, b_i)$, to check the correctness of parameters in $T_i \in T$ and (if accepted) to agree on these parameters, where $(\pi_i, r_i) \in \ddot{\pi}_i \in T_i$ and $1 \leq i \leq 2$. Note, if both \mathcal{B} and \mathcal{C} are honest, then $g_i = g'_i$. It also checks Δ in \mathcal{S} , e.g., to see whether it is sufficiently large.
 - if the above checks fail, it sets $a = 0$ and aborts. Otherwise, it sets $a = 1$. It sends a to \mathcal{S} .
4. Generating Update Request: $\text{genUpdateRequest}(T, f, \mathbf{l}) \rightarrow \hat{m}_1^{(C)}$.
Customer \mathcal{C} takes the following steps.
- sets its request parameter $m_1^{(C)}$ as below.
 - if it wants to set up a new payee, then it sets $m_1^{(C)} := (\phi, f)$, where f is the new payee's detail.
 - if it wants to amend the existing payee's detail, it sets $m_1^{(C)} := (i, f)$, where i is an index of the element in \mathbf{l} that should change to f .
 - at time t_0 , sends to \mathcal{S} the encryption of $m_1^{(C)}$, i.e., $\hat{m}_1^{(C)} = \text{Enc}(\bar{k}_1, m_1^{(C)})$.
5. Inserting New Payee: $\text{insertNewPayee}(\hat{m}_1^{(C)}, \mathbf{l}) \rightarrow \hat{\mathbf{l}}$.
Smart contract \mathcal{S} takes the following steps.
- if $\hat{m}_1^{(C)}$ is not empty, it appends $\hat{m}_1^{(C)}$ to the payee list $\hat{\mathbf{l}}$, resulting in an updated list, $\hat{\mathbf{l}}$.
 - if $\hat{m}_1^{(C)}$ is empty, it does nothing.
6. Generating Warning: $\text{genWarning}(T, \hat{\mathbf{l}}, aux) \rightarrow \hat{m}_1^{(B)}$.
Bank \mathcal{B} takes the following steps.
- checks if the most recent list $\hat{\mathbf{l}}$ is not empty. If it is empty, it halts. Otherwise, it proceeds to the next step.
 - decrypts each element of $\hat{\mathbf{l}}$ and checks its correctness, e.g., checks whether each element meets its internal policy stated in aux . If the check passes, it sets $m_1^{(B)} = \text{"pass"}$. Otherwise, it sets $m_1^{(B)} = \text{warning}$, where the warning is a string that contains a warning's detail concatenated with the string "warning".
 - at time t_1 , sends to \mathcal{S} the encryption of $m_1^{(B)}$, i.e., $\hat{m}_1^{(B)} = \text{Enc}(\bar{k}_1, m_1^{(B)})$.
7. Generating Payment Request: $\text{genPaymentRequest}(T, in_f, \hat{\mathbf{l}}, \hat{m}_1^{(B)}) \rightarrow \hat{m}_2^{(C)}$.
Customer \mathcal{C} takes the following steps.
- at time t_2 , decrypts $\hat{\mathbf{l}}$ and $\hat{m}_1^{(B)}$. Depending on the warning's content, it sets a payment request $m_2^{(C)}$ to ϕ or in_f , where in_f contains the payment's detail, e.g., the payee's detail in $\hat{\mathbf{l}}$ and the amount it wants to transfer.
 - at time t_3 , sends to \mathcal{S} the encryption of $m_2^{(C)}$, i.e., $\hat{m}_2^{(C)} = \text{Enc}(\bar{k}_1, m_2^{(C)})$.
8. Making Payment: $\text{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(B)}$.
Bank \mathcal{B} takes the following steps.
- at time t_4 , decrypts $\hat{m}_2^{(C)}$, i.e., $m_2^{(C)} = \text{Dec}(\bar{k}_1, \hat{m}_2^{(C)})$.
 - at time t_5 , checks the content of $m_2^{(C)}$. If $m_2^{(C)}$ is non-empty, i.e., $m_2^{(C)} = in_f$, it checks if the payee's detail in in_f has already been checked and the payment's amount does not exceed the customer's credit. If the checks pass, it runs the off-chain payment algorithm, $\text{pay}(in_f)$. In this case, it sets $m_2^{(B)} = \text{"paid"}$. Otherwise (i.e., if $m_2^{(C)} = \phi$ or neither checks pass), it sets $m_2^{(B)} = \phi$. It sends to \mathcal{S} the encryption of $m_2^{(B)}$, i.e., $\hat{m}_2^{(B)} = \text{Enc}(\bar{k}_1, m_2^{(B)})$.
9. Generating Complaint: $\text{genComplaint}(\hat{m}_1^{(B)}, \hat{m}_2^{(B)}, T, pk, aux_f) \rightarrow (\hat{z}, \hat{\pi})$.
Customer \mathcal{C} takes the following steps.
- decrypts $\hat{m}_1^{(B)}$ and $\hat{m}_2^{(B)}$; this results in $m_1^{(B)}$ and $m_2^{(B)}$ respectively. Depending on the content of the decrypted values, it sets its complaint's parameters $z := (z_1, z_2, z_3)$ as follows.
 - if \mathcal{C} wants to make one of the two below statements, it sets $z_1 = \text{"challenge message"}$.
 - the pass message (in $m_1^{(B)}$) should have been a warning.

- (ii) \mathcal{B} did not provide any message and if \mathcal{B} provided a warning, the fraud would have been prevented.
 - if \mathcal{C} wants to challenge the effectiveness of the warning (in $m_1^{(S)}$), it sets $z_2 = m || sig || pk_{\mathcal{G}}$ “challenge warning”, where m is a piece of evidence, $sig \in aux_f$ is the evidence’s certificate (obtained from \mathcal{G}), and $pk_{\mathcal{G}} \in pk$.
 - if \mathcal{C} wants to complain about the payment’s inconsistency, it sets $z_3 =$ “challenge payment”; else, it sets $z_3 = \phi$.
 - (b) at time t_6 , sends $\hat{z} = \text{Enc}(\bar{k}_1, z)$ and $\hat{\pi} = \tilde{\text{Enc}}(pk_{\mathcal{D}}, \tilde{\pi})$ to \mathcal{S} .
10. *Verifying Complaint*: $\text{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{m}, \hat{l}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{w}_j$.
Every $\mathcal{D}_j \in \{\mathcal{D}_1, \dots, \mathcal{D}_n\}$ acts as follows.
- (a) at time t_7 , decrypts $\hat{\pi}$, i.e., $\tilde{\pi} = \tilde{\text{Dec}}(sk_{\mathcal{D}}, \hat{\pi})$.
 - (b) checks the validity of $(\tilde{\pi}_1, \tilde{\pi}_2)$ in $\tilde{\pi}$ by locally running the SAP’s verification, i.e., $\text{SAP.verify}(\cdot)$, for each $\tilde{\pi}_i$. It returns s . If $s = 0$, it halts. If $s = 1$ for both $\tilde{\pi}_1$ and $\tilde{\pi}_2$, it proceeds to the next step.
 - (c) decrypts $\hat{m} = [\hat{m}_1^{(C)}, \hat{m}_2^{(C)}, \hat{m}_1^{(S)}, \hat{m}_2^{(S)}]$ using $\text{Dec}(\bar{k}_1, \cdot)$, where $\bar{k}_1 \in \tilde{\pi}_1$. Let $[m_1^{(C)}, m_2^{(C)}, m_1^{(S)}, m_2^{(S)}]$ be the result.
 - (d) checks whether \mathcal{C} made an update request to its payee’s list. To do so, it checks if $m_1^{(C)}$ is non-empty and (its encryption) was registered by \mathcal{C} in \mathcal{S} . Also, it checks whether \mathcal{C} made a payment request, by checking if $m_2^{(C)}$ is non-empty and (its encryption) was registered by \mathcal{C} in \mathcal{S} at time t_3 . If either check fails, it halts.
 - (e) decrypts \hat{z} and \hat{l} using $\text{Dec}(\bar{k}_1, \cdot)$, where $\bar{k}_1 \in \tilde{\pi}_1$. Let $z := (z_1, z_2, z_3)$ and l be the result.
 - (f) sets its verdicts according to the content of $z := (z_1, z_2, z_3)$, as follows.
 - if “challenge message” $\notin z_1$, it sets $w_{1,j} = 0$. Otherwise, it runs $\text{verStat}(add_{\mathcal{S}}, m_1^{(S)}, l, \Delta, aux) \rightarrow w_{1,j}$, to determine if a warning (in $m_1^{(S)}$) should have been given (instead of the pass or no message).
 - if “challenge warning” $\notin z_2$, it sets $w_{2,j} = w_{3,j} = 0$. Otherwise, it runs $\text{checkWarning}(add_{\mathcal{S}}, z_2, m_1^{(S)}, aux') \rightarrow (w_{2,j}, w_{3,j})$, to determine the effectiveness of the warning (in $m_1^{(S)}$).
 - if “challenge payment” $\in z_3$, it checks if the payment was made. If it passes, it sets $w_{4,j} = 1$. If it fails, it sets $w_{4,j} = 0$. If “challenge payment” $\notin z_3$, it checks if “paid” $\in m_2^{(C)}$. If it passes, it sets $w_{4,j} = 1$. Else, it sets $w_{4,j} = 0$.
 - (g) encodes its verdicts $(w_{1,j}, w_{2,j}, w_{3,j}, w_{4,j})$ as follows.
 - i. locally maintains a counter, o_{adr_c} , for each \mathcal{C} . It sets its initial value to 0.
 - ii. calls $\text{PVE}(\cdot)$ to encode each verdict. In particular, it performs as follows. $\forall i, 1 \leq i \leq 4$:
 - calls $\text{PVE}(\bar{k}_0, adr_c, w_{i,j}, o_{adr_c}, n, j) \rightarrow \bar{w}_{i,j}$.
 - sets $o_{adr_c} = o_{adr_c} + 1$.

By the end of this step, a vector \bar{w}_j of four encoded verdicts is computed, i.e., $\bar{w}_j = [\bar{w}_{1,j}, \dots, \bar{w}_{4,j}]$.

 - iii. uses $\bar{k}_2 \in \tilde{\pi}_2$ to further encode/encrypt $\text{PVE}(\cdot)$ ’s outputs as follows. $\hat{w}_j = \text{Enc}(\bar{k}_2, \bar{w}_j)$.
 - (h) at time t_8 , sends to \mathcal{S} the encrypted vector, \hat{w}_j .
11. *Resolving Dispute*: $\text{resDispute}(T_2, \hat{w}, pp) \rightarrow v$.
Party \mathcal{DR} takes the below steps at time t_9 , if it is invoked by \mathcal{C} or \mathcal{S} which sends $\tilde{\pi}_2 \in T_2$ to it.
- (a) checks $\tilde{\pi}_2$ ’s validity by locally running the SAP’s verification, i.e., $\text{SAP.verify}(\cdot)$, that returns s . If $s = 0$, it halts.
 - (b) computes the final verdicts, as below.
 - i. uses $\bar{k}_2 \in \tilde{\pi}_2$ to decrypt the auditors’ encoded verdicts, as follows. $\forall j, 1 \leq j \leq n : \bar{w}_j = \text{Dec}(\bar{k}_2, \hat{w}_j)$, where $\hat{w}_j \in \hat{w}$.
 - ii. constructs four vectors, $[u_1, \dots, u_4]$, and sets each vector u_i as follows. $\forall i, 1 \leq i \leq 4 : u_i = [\bar{w}_{i,1}, \dots, \bar{w}_{i,n}]$, where $\bar{w}_{i,j} \in \bar{w}_j$.
 - iii. calls $\text{FVD}(\cdot)$ to extract each final verdict, as follows. $\forall i, 1 \leq i \leq 4 : \text{calls } \text{FVD}(n, u_i) \rightarrow v_i$.
 - (c) outputs $v = [v_1, \dots, v_4]$.

Customer \mathcal{C} must be reimbursed if the final verdict is that (i) the “pass” message or missing message should have been a warning or (ii) the warning was ineffective and the provided evidence was not invalid, and (iii) the payment has been made. Stated formally, the following relation must hold:

$$\left(\underbrace{(v_1 = 1)}_{(i)} \vee \underbrace{(v_2 = 1 \wedge v_3 = 1)}_{(ii)} \right) \wedge \left(\underbrace{v_4 = 1}_{(iii)} \right)$$

Note, in the above PwDR protocol, even \mathcal{C} and \mathcal{B} that know the decryption secret keys, (\bar{k}_1, \bar{k}_2) , cannot link a certain verdict to an auditor, for two main reasons; namely, (a) they do not know the masking random values used by auditors to mask each verdict and (b) the final verdicts (v_1, \dots, v_4) reveal nothing about the number of 1 or 0 verdicts, except when all auditors vote 0. In the PwDR, we used PVE and FVD only because they are highly efficient. However, it is easy to replace them with GPVE and GFVD.

Theorem 3. *The above PwDR protocol is secure, with regard to definition 7, if the digital signature is existentially unforgeable under chosen message attacks, the blockchain, SAP, and pseudorandom function $\text{PRF}(\cdot)$ are secure, the encryption schemes are semantically secure, and the correctness of verdict encoding-decoding protocols (i.e., PVE and FVD) holds.*

7 Security Analysis of the PwDR Protocol

In this section, we present the proof sketch of the PwDR protocol's security, i.e., Theorem 3. To prove the theorem, we show that the PwDR satisfies all security properties defined in Section 5. We first prove that it meets security against a malicious victim.

Lemma 1. *If the digital signature is existentially unforgeable under chosen message attacks, and the SAP and blockchain are secure, then the PwDR scheme is secure against a malicious victim, with regard to Definition 4.*

Proof. First, we focus on event I : $\left((m_1^{(\mathcal{B})} = \text{warning}) \wedge \left(\sum_{j=1}^n w_{1,j} \geq e \right) \right)$ which considers the case where \mathcal{B} has provided a warning message but \mathcal{C} manages to convince at least the threshold of the auditors to set their verdicts to 1, that ultimately results in $\sum_{j=1}^n w_{1,j} \geq e$. We argue that the adversary's success probability in this event is negligible in the security parameter. Specifically, due to the security of SAP, \mathcal{C} cannot convince an auditor to accept a different decryption key, e.g., $k' \in \tilde{\pi}'$, that will be used to decrypt \mathcal{B} 's encrypted message $\hat{m}_1^{(\mathcal{B})}$, other than what was agreed between \mathcal{C} and \mathcal{B} in the initiation phase, i.e., $\bar{k}_1 \in \tilde{\pi}_1$. To be more precise, it cannot persuade an auditor to accept a statement $\tilde{\pi}'$, where $\tilde{\pi}' \neq \tilde{\pi}_1$, except with a negligible probability, $\mu(\lambda)$. This ensures that honest \mathcal{B} 's original message (and accordingly the warning) is accessed by every auditor with a high probability. Next, we consider event II : $\left(\left(\sum_{j=1}^n w_{1,j} < e \right) \wedge (v_1 = 1) \right)$ that captures the case where only less than the threshold of the auditors approved that the pass message was given incorrectly or the missing message could prevent the APP fraud, but the final verdict that \mathcal{DR} extracts implies that at least the threshold of the auditors approved that. We argue that the probability that this event occurs is negligible in the security parameter too. Specifically, due to the security of the SAP, \mathcal{C} cannot persuade (a) an auditor to accept a different encryption key and (b) \mathcal{DR} to accept a different decryption key other than what was agreed between \mathcal{C} and \mathcal{B} in the initiation phase. More precisely, it cannot persuade them to accept a statement $\tilde{\pi}'$, where $\tilde{\pi}' \neq \tilde{\pi}_2$, except with a negligible probability, $\mu(\lambda)$.

Now, we move on to event III : $\left((\text{checkWarning}(m_1^{(\mathcal{B})}) = 1) \wedge \left(\sum_{j=1}^n w_{2,j} \geq e \right) \right)$. It captures the case where \mathcal{B} has provided an effective warning message but \mathcal{C} manages to make at least the threshold of the auditors set their verdicts to 1, that ultimately results in $\sum_{j=1}^n w_{2,j} \geq e$. The same argument provided to event I is applicable to this even too. Briefly, due to the security of the SAP, \mathcal{C} cannot persuade an auditor to accept a different decryption key other than what was agreed between \mathcal{C} and \mathcal{B} in the initiation phase. Therefore, all auditors will receive the original message of \mathcal{B} , including the effective warning message, except a negligible probability, $\mu(\lambda)$. Now, we consider event IV : $\left(\left(\sum_{j=1}^n w_{2,j} < e \right) \wedge (v_2 = 1) \right)$ which captures the

case where at least the threshold of the auditors approved that the warning message was effective but the final verdict that \mathcal{DR} extracts implies that they approved the opposite. The security argument of event II applies to this event as well. In short, due to the security of the SAP, \mathcal{C} cannot persuade an auditor to accept a different encryption key, and cannot convince \mathcal{DR} to accept a different decryption key other than what was initially agreed between \mathcal{C} and \mathcal{B} , except a negligible probability, $\mu(\lambda)$. Now, we analyse event $V : \left(u \notin Q \wedge \text{Sig.ver}(pk, u, sig) = 1 \right)$. This even captures the case where the malicious victim comes up with a valid signature/certificate on a message that has never been queried to the signing oracle. Nevertheless, due to the existential unforgeability of the digital signature scheme, the probability that such an event occurs is negligible, $\mu(\lambda)$. Next, we focus on event $VI : \left(\left(\sum_{j=1}^n w_{3,j} < e \right) \wedge (v_3 = 1) \right)$ that considers the case where less than the threshold of the auditors indicated that the signature (in \mathcal{C} 's complaint) is valid, but the final verdict that \mathcal{DR} extracts implies that at least the threshold of the auditors approved the signature. This means the adversary has managed to switch the verdicts of those auditors who voted 0 to 1. However, the probability that this even occurs is negligible as well. Because, due to the SAP's security, \mathcal{C} cannot convince an auditor and \mathcal{DR} to accept different encryption and decryption keys other than what was initially agreed between \mathcal{C} and \mathcal{B} , except a negligible probability, $\mu(\lambda)$. Therefore, with only a negligible probability the adversary can switch a verdict for 0 to the verdict for 1.

Moreover, a malicious \mathcal{C} cannot frame an honest \mathcal{B} for providing an invalid message by manipulating the smart contract's content, e.g., by replacing an effective warning with an ineffective one in \mathcal{S} , or excluding a warning from \mathcal{S} . To do that, it has to either forge the honest party's signature, so it can send an invalid message on its behalf, or fork the blockchain so the chain comprising a valid message is discarded. In the former case, the adversary's probability of success is negligible as long as the signature is secure. The adversary has the same success probability in the latter case because it has to generate a long enough chain that excludes the valid message which has a negligible success probability, under the assumption that the hash power of the adversary is lower than those of honest miners and due to the blockchain's liveness property an honestly generated transaction will eventually appear on an honest miner's chain [21]. \square

Now, we first present a lemma formally stating that the PwDR protocol is secure against a malicious bank and then prove this lemma.

Lemma 2. *If the SAP and blockchain are secure, and the correctness of verdict encoding-decoding protocols (i.e., PVE and FVD) holds, then the PwDR protocol is secure against a malicious bank, with regard to Definition 5.*

Proof. We first focus on event $I : \left(\left(\sum_{j=1}^n w_{1,j} \geq e \right) \wedge (v_1 = 0) \right)$ which captures the case where \mathcal{DR} is convinced that the pass message was correctly given or an important warning message was not missing, despite at least the threshold of the auditors do not believe so. We argue that the probability that this event takes place is negligible in the security parameter. Because \mathcal{B} cannot persuade \mathcal{DR} to accept a different decryption key, e.g., $k' \in \bar{\pi}'$, other than what was agreed between \mathcal{C} and \mathcal{B} in the initiation phase, i.e., $\bar{k}_2 \in \bar{\pi}_2$, except with a negligible probability. Specifically, it cannot persuade \mathcal{DR} to accept a statement $\bar{\pi}'$, where $\bar{\pi}' \neq \bar{\pi}_2$ except with probability $\mu(\lambda)$. Also, as discussed in Section 6.4, due to the correctness of the verdict encoding-decoding protocols, i.e., PVE and FVD, the probability that multiple representations of verdict 1 cancel out each other is negligible too, i.e., it is at most $\frac{1}{2^\lambda}$. Thus, event I occurs only with a negligible probability, $\mu(\lambda)$. To assert that events $II : \left(\left(\sum_{j=1}^n w_{2,j} \geq e \right) \wedge (v_2 = 0) \right)$, $III : \left(\left(\sum_{j=1}^n w_{3,j} \geq e \right) \wedge (v_3 = 0) \right)$, and $IV : \left(\left(\sum_{j=1}^n w_{4,j} \geq e \right) \wedge (v_4 = 0) \right)$ occur only with a negligible probability, we can directly use the above argument provided for event I. To avoid repetition, we do not restate them in this proof. Moreover, a malicious \mathcal{B} cannot frame an honest \mathcal{C} for providing an invalid message by manipulating the smart contract's content, e.g., by replacing its valid signature with an invalid one or sending a message on its behalf, due to the security of the blockchain. \square

Next, we prove the PwDR protocol's privacy. As before, we first formally state the related lemma and then prove it.

Lemma 3. *If the encryption schemes are semantically secure, and the SAP and encoding-decoding schemes (i.e., PVE and FVD) are secure, then the PwDR protocol is privacy-preserving with regard to Definition 6.*

Proof. We first focus on property 1, i.e., the privacy of the parties' messages from the public. Due to the privacy-preserving property of the SAP, that relies on the hiding property of the commitment scheme, given the public commitments, $g := (g_1, g_2)$, the adversary learns no information about the committed values, (\bar{k}_1, \bar{k}_2) , except with a negligible probability, $\mu(\lambda)$. Thus, it cannot find the encryption-decryption keys used to generate ciphertext $\hat{\mathbf{m}}, \hat{\mathbf{l}}, \hat{z}$, and $\hat{\mathbf{w}}$. Moreover, due to the semantical security of the symmetric key and asymmetric key encryption schemes, given ciphertext $(\hat{\mathbf{m}}, \hat{\mathbf{l}}, \hat{z}, \hat{\pi}, \hat{\mathbf{w}})$ the adversary cannot learn anything about the related plaintext, except with a negligible probability, $\mu(\lambda)$. Thus, in experiment $\text{Exp}_3^{\mathcal{A}_1}$, adversary \mathcal{A}_1 cannot tell the value of $\gamma \in \{0, 1\}$ significantly better than just guessing it, i.e., its success probability is at most $\frac{1}{2} + \mu(\lambda)$. Now we move on to property 2, i.e., the privacy of each verdict from \mathcal{DR} . Due to the privacy-preserving property of the SAP, given $g_1 \in g$, a corrupt \mathcal{DR} cannot learn \bar{k}_1 . So, it cannot find the encryption-decryption key used to generate ciphertext $\hat{\mathbf{m}}, \hat{\mathbf{l}}$, and \hat{z} . Also, public parameters (pk, pp) and token T_2 are independent of \mathcal{C} 's and \mathcal{B} 's exchanged messages (e.g., payment requests or warning messages) and \mathcal{D}_j 's verdicts. Furthermore, due to the semantical security of the symmetric key and asymmetric key encryption schemes, given ciphertext $(\hat{\mathbf{m}}, \hat{\mathbf{l}}, \hat{z}, \hat{\pi})$ the adversary cannot learn anything about the related plaintext, except with a negligible probability, $\mu(\lambda)$. Also, due to the security of the PVE and FVD protocols, the adversary cannot link a verdict to a specific auditor with a probability significantly better than the maximum probability, Pr' , that an auditor sets its verdict to a certain value, i.e., its success probability is at most $Pr' + \mu(\lambda)$, even if it is given the final verdicts, except when all auditors' verdicts are 0. We conclude that, excluding the case where all verdicts are 0, given $(T_2, pk, pp, g, \hat{\mathbf{m}}, \hat{\mathbf{l}}, \hat{z}, \hat{\pi}, \hat{\mathbf{w}}, \mathbf{v})$, adversary \mathcal{A}_3 's success probability in experiment $\text{Exp}_4^{\mathcal{A}_2}$ to link a verdict to an auditor is at most $Pr' + \mu(\lambda)$. \square

Theorem 4. *The PwDR protocol is secure according to Definition 7.*

Proof. Due to Lemma 1, the PwDR protocol is secure against a malicious victim. Also, due to lemmas 2 and 3 it is secure against a malicious bank and is privacy-preserving, respectively. Thus, it satisfies all the properties of Definition 7, meaning that the PwDR protocol is indeed secure according to this definition. \square

8 Evaluation

8.1 Asymptotic Cost Analysis

In this section, we analyse the PwDR protocol's computation and communication asymptotic cost. Table 2 summarises the analysis result.

Table 2: The PwDR's asymptotic cost. In the table, n is the number of auditors and e is the threshold.

| Party | Setting | | Computation Cost | Communication Cost |
|---|---------|---------|---------------------------------------|---------------------------------------|
| | $e = 1$ | $e > 1$ | | |
| Customer \mathcal{C} | ✓ | ✓ | $O(1)$ | $O(1)$ |
| Bank \mathcal{B} | ✓ | ✓ | $O(1)$ | $O(1)$ |
| Auditor $\mathcal{D}_1, \dots, \mathcal{D}_{n-1}$ | ✓ | ✓ | $O(1)$ | $O(1)$ |
| Auditor \mathcal{D}_n | ✓ | | $O(n)$ | $O(1)$ |
| | | ✓ | $O(\sum_{i=e}^n \frac{n!}{i!(n-i)!})$ | $O(\sum_{i=e}^n \frac{n!}{i!(n-i)!})$ |
| Dispute resolver \mathcal{DR} | ✓ | ✓ | $O(n)$ | $O(1)$ |

Computation Cost. We first analyse \mathcal{C} 's cost. In Phase 3, \mathcal{C} invokes a hash function twice to check the correctness of the private statements' parameters. In Phase 4, it invokes the symmetric encryption once to encrypt its update request. In Phase 7, it invokes the symmetric encryption twice to decrypt \mathcal{B} 's warning message and to encrypt its payment request. In Phase 9, it runs the symmetric encryption three times to decrypt \mathcal{B} 's warning and payment messages and to encrypt its complaint. In the same phase, it invokes the asymmetric encryption once to encrypt the private statements' opening. Therefore, \mathcal{C} 's complexity is constant, $O(1)$, i.e., independent of n . Next, we analyse \mathcal{B} 's cost. In Phase 2, it invokes the hash function twice to commit to two statements. In Phase 6, it calls the symmetric key encryption once to encrypt its outgoing warning message. In Phase 8, it also invokes the symmetric key encryption once to encrypt the outgoing payment message. Thus, \mathcal{B} 's complexity is $O(1)$ too. Next, we analyse each auditor's cost. In Phase 10, each \mathcal{D}_j invokes the asymmetric key encryption once to decrypt the private statements' openings. It also invokes the hash function twice to verify the openings. It invokes the symmetric key encryption six times to decrypt \mathcal{C} 's and \mathcal{B} 's messages that were posted on \mathcal{S} (this includes \mathcal{C} 's complaint). Recall, in the same phase, each auditor encodes its verdict using a verdict encoding protocol. Now, we evaluate the verdict encoding complexity of each auditor for two cases: (a) $e = 1$ and (b) $e \in (1, n]$. Note, in the former case the PVE is invoked while in the latter GPVE is invoked. In case (a), every auditor \mathcal{D}_j , except \mathcal{D}_n , invokes the pseudorandom function once to encode its verdict. However, auditor \mathcal{D}_n invokes the pseudorandom function $n - 1$ times and XORs the function's outputs with each other. Thus, in case (a), auditor \mathcal{D}_n 's complexity is $O(n)$ while the rest of auditors' complexity is $O(1)$. In case (b), every auditor \mathcal{D}_j , except \mathcal{D}_n , invokes the pseudorandom function twice to encode its verdict. But, auditor \mathcal{D}_n invokes the pseudorandom function $n - 1$ times and XORs the function's outputs with each other. It also invokes the pseudorandom function n times to generate all auditors' representations of verdict 1. It computes all $y = \sum_{i=e}^n \frac{n!}{i!(n-i)!}$ combinations of the representations that meet the threshold which involves $O(y)$ XORs. It also inserts y elements into a Bloom filter that requires $O(y)$ hash function evaluations. So, in case (b), auditor \mathcal{D}_n 's complexity is $O(y)$ while the rest of the auditors' complexity is $O(1)$. To conclude, in Phase 10, auditor \mathcal{D}_n 's complexity is either $O(n)$ or $O(y)$, while the rest of the auditors' complexity is $O(1)$. Now, we analyse \mathcal{DR} 's cost in Phase 11. It invokes the hash function once to check the private statement's correctness. It also performs $O(n)$ symmetric key decryption to decrypt auditors' encoded verdicts. Now, we evaluate the verdict decoding complexity of \mathcal{DR} for two cases: (a) $e = 1$ and (b) $e \in (1, n]$. In the former case (in which FVD is invoked), it performs $O(n)$ XOR to combine all verdicts. Its complexity is also $O(n)$ in the latter case (in which GFVD is invoked), with a small difference that it also invokes the Bloom filter's hash functions, to make a membership query to the Bloom filter. Thus, \mathcal{DR} 's complexity is $O(n)$.

Communication Cost. Now, we analyse the communication cost of the PwDR protocol. Briefly, \mathcal{C} 's complexity is $O(1)$ as in total it sends only six messages to other parties. Similarly, \mathcal{B} 's complexity is $O(1)$ as its total number of outgoing messages is only nine. Each auditor \mathcal{D}_j sends only four messages to the smart contract, so its complexity is $O(1)$. However, if GFVD is invoked, then auditor \mathcal{D}_n needs to send also a Bloom filter that costs it $O(y)$. Moreover, \mathcal{DR} 's complexity is $O(1)$, as its outgoing messages include only four binary values.

8.2 Concrete Performance Analysis

In this section, we study the PwDR protocol's concrete performance. As we saw in the previous section, the customer's and bank's complexity is very low and constant; however, one of the auditors, i.e., auditor \mathcal{D}_n , and the dispute resolver have non-constant complexities. These non-constant overheads were mainly imposed by the verdict inducing-decoding protocols. Therefore, to study these parties' runtime in the PwDR, we implemented both variants of the verdict encoding-decoding protocols (that were presented in Section 6.4). They were implemented in C++. The source code of variants 1 and 2 is available in [2] and [3] respectively. To conduct the experiment, we used a MacBook Pro laptop with quad-core Intel Core i5, 2 GHz CPU, and 16 GB RAM. We ran the experiment on average 100 times. The prototype implementation uses the

“Cryptopp” library³ for cryptographic primitives, the “GMP” library⁴ for arbitrary precision arithmetic, and the “Bloom Filter” library⁵. In the experiment, we set the false-positive rate in a Bloom filter to 2^{-40} and the finite field size to 128 bits. Table 3 presents \mathcal{D}_n ’s and \mathcal{DR} ’s runtime for various numbers of auditors in two cases; namely, when the threshold is 1 and when it is greater than 1. In the former case, we used the PVE and FVD protocols. In the latter case, we used the GPVE and GFVD ones.

Table 3: The PwDR’s runtime (in ms). Broken-down by parties. In the table, n is the number of arbiters and e is the threshold.

| Party | $n = 6$ | | $n = 8$ | | $n = 10$ | | $n = 12$ | |
|---------------------------------|---------|---------|---------|---------|----------|---------|----------|---------|
| | $e = 1$ | $e = 4$ | $e = 1$ | $e = 5$ | $e = 1$ | $e = 6$ | $e = 1$ | $e = 7$ |
| Arbiter \mathcal{D}_n | 0.019 | 0.220 | 0.033 | 0.661 | 0.035 | 2.87 | 0.052 | 10.15 |
| Dispute resolver \mathcal{DR} | 0.001 | 0.015 | 0.001 | 0.016 | 0.001 | 0.069 | 0.003 | 0.09 |

As the table depicts, the runtime of \mathcal{D}_n increases gradually from 0.019 to 10.15 milliseconds when the number of auditors grows from $n = 6$ to $n = 12$. In contrast, the runtime of \mathcal{DR} grows slower; it increases from 0.001 to 0.09 milliseconds when the number of auditors increases. Nevertheless, the overall cost is very low. Specifically, the highest runtime is only about 10 milliseconds which belongs to \mathcal{D}_n when $n = 12$ and $e = 7$. It is also evident that the parties’ runtime in the PVE and FVD protocols is much lower than their runtime in the GPVE and GFVD ones. To compare the parties’ runtime, we also fixed the threshold to 6 (in GPVE and GFVD protocols) and ran the experiment for different values of n . Figure 7 summarises the result. As this figure indicates, the runtime of \mathcal{D}_n and \mathcal{DR} almost linearly grows when the number of auditors increases. Moreover, \mathcal{D}_n has a higher runtime than \mathcal{DR} has, and its runtime growth is faster than that of \mathcal{DR} .

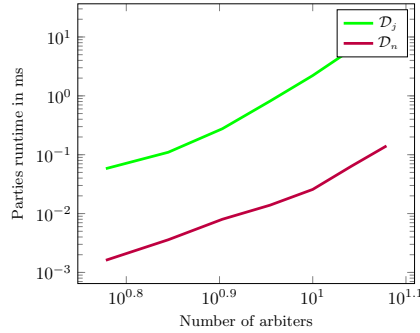


Fig. 7: Parties’ runtime in the PwDR.

9 Future Research

In this section, we highlight a set of future research directions in the context of APP frauds.

³ <https://www.cryptopp.com>

⁴ <https://gmplib.org>

⁵ <http://www.partow.net/programming/bloomfilter/index.html>

9.1 Improving Warnings’ Effectiveness

As we stated in Section 4.2, one of the determining factors in the process of allocating liability to APP fraud victims is if they follow warnings. However, there exists no publicly available study on the effectiveness of warnings in the context of APP frauds. There exists a comprehensive research line in determining the effectiveness of warnings in general, e.g., in [13,19,26]. These traditional research line studies which factors make warnings effective and how a warning recipient is attracted to and follows the warning message. Nevertheless, in the context of APP frauds, there is a vital unique factor that can directly influence a warning’s effectiveness. The factor is the ability of fraudsters to interact directly with their victims. This lets a fraudster actively try to negate the effectiveness of a bank’s warning and persuade the warning recipient to ignore the warning (and make payment). Such a factor was not (needed to be) taken into account in the traditional study of warnings. The current high rate of APP frauds occurrence suggests that there exists a huge room for improving warnings’ effectiveness. Thus, future research can investigate (via users studies) and identify key factors that can improve the effectiveness of warnings in this context.

9.2 Protecting APP Fraud Victims of Alternative Payment Platforms

To date, there is no official report on the occurrence of APP frauds on any payment platforms (e.g., cryptocurrency) other than regular banking. This can be due to a lack of an oversight organisation which collects fraud-related data or due to a low rate of such frauds taking place in these platforms because these platforms are not popular enough. Nevertheless, with the increase in the popularity of alternative payment platforms, including “Central Bank Digital Currency”(CBDC), it is likely that APP fraudsters will target these platforms’ users. Hence, another interesting future research direction would be to design secure dispute resolution protocols to protect APP frauds victims in these platforms as well.

9.3 Studying Users’ Compliance with the CRM Code’s Guidelines

Currently, customers are expected to comply with the CRM code’s guidelines. Users’ compliance with these guidelines could help lower the rate of APP frauds occurrence. Also, if victims fail to comply with such guidelines, then banks can claim that customers’ have been negligent which ultimately could cost the victims. Such guidelines would be effective when they are known and followed by customers. Recently, Van Der Zee [44] has conducted an interesting study to find out whether customers of the “Dutch Banking Association” (in the Netherlands) are aware of the bank’s digital payments guidelines and if so, whether they comply with these guidelines. But, there exists no systematic study to investigate whether customers are aware of and comply with the CRM code’s guidelines. Therefore, another research direction is to fill the above void.

9.4 Ensuring Security Against Exploitative Victims

Having in place a transparent deterministic procedure (e.g., the PwDR protocol) for evaluating victims’ requests for reimbursement could potentially create opportunities for exploitation. In particular, an honest victim of an APP fraud that had been reimbursed in the past due to the payment system’s vulnerability (e.g., an ineffective warning) may be tempted to exploit the same known vulnerability multiple times. Hence, future research can investigate how to secure the online banking system against such exploitative victims.

10 Related Work

10.1 Authorised Push Payment Fraud

Anderson *et al.* [6] provide an overview of APP frauds and highlight that although the (CRM) code would urge banks to accept more liability for APP frauds, it remains to be seen how this will evolve as fraudsters will continuously try to figure out how bank systems can facilitate misdirection attacks. Taylor *et al.* [35] analyses

the CRM code from legal and practical perspectives. They state that this code’s proper implementation would make considerable advances to protect victims of APP frauds. They also highlight this code’s shortcomings and argue that the code is still ambiguous. Specifically, they refer to a part of the CRM code stating that for a victim to be liable for reimbursement, it needs to take an appropriate level of care when an APP fraud has been taking place, and argue that this code does not clarify (i) the level of care needed, and (ii) how to verify that the victim has taken adequate steps. Our PwDR can be considered as a remedy for the above issues. Moreover, Kjørven [24] investigates whether banks or customers should be liable for customers’ financial loss to online frauds including APP ones, under Scandinavian and European law. The author states that consumers are often left to deal with the losses caused by APP frauds. She concludes that this should change and a larger portion of the losses should be allocated to financial institutions.

10.2 Dispute Resolution

In payment platforms, dispute resolution approaches can be broadly categorised into two classes, (a) *centralised* and (b) *decentralised*. In the former class, at any point in time, a single party tries to settle the dispute. In particular, if a customer disputes having made or authorised a transaction, then the related bank tries directly settle the dispute with the customer. However, the banks’ terms and conditions can complicate the dispute resolution process. In 2000, Bohm *et al.* [11] analysed different terms of banks in the UK. They argued that the approach taken by banks is unfair to their customers in some cases. Later, Anderson [7] points out that the move to online banking led many financial institutions to impose terms and conditions on their customers that ultimately would shift the burden of proof in dispute to the customer (who often cannot provide the required proof due to lack of training or related skills). In the UK, if banks and customers cannot settle a dispute directly, upon the customers’ request, the Financial Ombudsman Service would step in to resolve the dispute. If the customer is not satisfied with the decision of this third party, it can take a step further and take the matter to court, which will be a tedious process. Later, Becker *et al.* [9] investigate to what extent bank customers know the terms and conditions (T&C) they signed up. Their study suggests that only 35% fully understand T&C and 28% find important parts of T&C are unclear. They also point out that payment regulations in the US are more customer-friendly and mostly are in customers’ favor in disputes than those regulations in the EU and UK.

Now we turn our attention to the latter class, i.e., decentralised dispute resolution. After the invention of the (decentralised) blockchain technology and especially its vital side-product, smart contract, researchers considered the possibility of resolving disputes in a decentralised manner by relying on smart contracts. Such a possibility has been discussed and studied by the law research community, e.g., in [14,30,31]. Moreover, various ad-hoc blockchain-based cryptographic protocols have been proposed to resolve disputes in different contexts and settings. We briefly explain a few of them. Dziembowski *et al.* [16] propose FairSwap, an efficient protocol that allows a seller and buyer to fairly exchange digital items and coins. It is mainly based on a Merkle tree and Ethereum smart contracts which can efficiently resolve a dispute between the seller and buyer when the two parties disagree. Recently, researchers in [17] propose OPTISWAP that improves FairSwap’s performance. Similar to FairSwap, OPTISWAP uses a Merkle tree and smart contract, but it relies on an *interactive* dispute resolution protocol. Recently, Abadi *et al.* [5] propose a privacy-preserving protocol that allows a fair exchange of digital coins and a certain *digital service*, called “proofs of data retrievability”. To settle disputes between the seller and buyer efficiently, the protocol uses a combination of blockchain, modified Merkle tree, symmetric key encryption, and a third-party arbiter. In the context of verifiable (cloud) computation, Dong *et al.* [15] use a combination of smart contracts, game theory (incentivization), and a third-party arbiter to design an efficient protocol that lets a client outsource its expensive computation to the cloud servers such that it can efficiently check the result’s correctness. In the case of dispute, the protocol lets the parties invoke the arbiter which efficiently settles the dispute with the assistance of the smart contract. Green *et al.* [22] propose a variant of payment channel [33] (which improves cryptocurrencies’ scalability) while preserving the users’ anonymity. In this scheme, in the case of a dispute between two parties, they can send a set of proofs to a smart contract that settles the disputes between the two.

Although various dispute resolution solutions have been proposed, to date, there exists no secure (centralised or decentralised) solution designed to resolve disputes in the context of APP frauds. Our PwDR is the first protocol that fills the gap.

11 Conclusion

An APP fraud takes place when fraudsters deceive a victim to make a payment to a bank account controlled by the fraudsters. Although APP frauds have been growing at a concerning rate, the victims are not receiving a sufficient level of protection and the reimbursement rate is still low. Authorities and regulators have provided guidelines to prevent APP frauds occurrence and improve victims' protection, but these guidelines are still vague and open to interpretation. In this work, to facilitate APP frauds victims' reimbursement, we proposed the notion of payment with dispute resolution. We identified the vital properties that such a notion should possess and formally defined them. We also proposed a candidate construction, PwDR, and proved its security. The PwDR not only offers transparency and accountability but also acts as a data hub providing sufficient information that could help regulators examine whether the reimbursement regulations have been applied correctly and consistently among financial institutions. We also studied the PwDR's cost via asymptotic and concrete runtime evaluation. Our cost analysis indicated that the construction is efficient.

References

1. A P20 Fraud and Criminal Transactions Working Group Paper: Best practice approaches for combating payee scams (2021), <https://static1.squarespace.com/static/5efcc6dae323db37b4d01d19/t/60958e7453a1e0728b445470/1620414069082/P20+Report+-+Best+Practice+Approaches+For+Combating+Payee+Scams.pdf>
2. Abadi, A.: Variant 1: Efficient verdict encoding-decoding protocol (2021), <https://github.com/AydinAbadi/PwDR/blob/main/PwDR-code/encoding-decoding.cpp>
3. Abadi, A.: Variant 2: Generic verdict encoding-decoding protocol (2021), <https://github.com/AydinAbadi/PwDR/blob/main/PwDR-code/generic-encoding-decoding.cpp>
4. Abadi, A., Murdoch, S.J., Zacharias, T.: Recurring contingent payment for proofs of retrievability. Cryptology ePrint Archive, Report 2021/1145 (2021), <https://ia.cr/2021/1145>
5. Abadi, A., Murdoch, S.J., Zacharias, T.: Recurring contingent payment for proofs of retrievability. IACR Cryptol. ePrint Arch. (2021)
6. Anderson, R., Barton, C., Bölme, R., Clayton, R., Ganán, C., Grasso, T., Levi, M., Moore, T., Vasek, M.: Measuring the changing cost of cybercrime (2019)
7. Anderson, R., et al.: Closing the phishing hole—fraud, risk and nonbanks. In: Federal Reserve Bank of Kansas City—Payment System Research Conferences. pp. 41–56 (2007)
8. Authority, F.C.: FCA glossary (2021), <https://www.handbook.fca.org.uk/handbook/glossary/G3566a.html>
9. Becker, I., Hutchings, A., Abu-Salma, R., Anderson, R.J., Bohm, N., Murdoch, S.J., Sasse, M.A., Stringhini, G.: International comparison of bank fraud reimbursement: customer perceptions and contractual terms. J. Cybersecur. (2017)
10. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. (1970)
11. Bohm, N., Brown, I., Gladman, B.: Electronic commerce: Who carries the risk of fraud? J. Inf. Law Technol. 2000 (2000)
12. Bose, P., Guo, H., Kranakis, E., Maheshwari, A., Morin, P., Morrison, J., Smid, M.H.M., Tang, Y.: On the false-positive rate of bloom filters. Inf. Process. Lett. (2008)
13. Brinton Anderson, B., Vance, A., Kirwan, C.B., Eargle, D., Jenkins, J.L.: How users perceive and respond to security messages: a neurois research agenda and empirical study. European Journal of Information Systems 25(4), 364–390 (2016)
14. Buchwald, M.: Smart contract dispute resolution: the inescapable flaws of blockchain-based arbitration. U. Pa. L. Rev. (2019)
15. Dong, C., Wang, Y., Aldweesh, A., McCorry, P., van Moorsel, A.: Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In: CCS (2017)
16. Dziembowski, S., Ekey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: CCS (2018)

17. Ecekey, L., Faust, S., Schlosser, B.: Optiswap: Fast optimistic fair exchange. In: ASIA CCS (2020)
18. Federal Bureau of Investigation (FBI): Internet crime report (2020), https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf
19. Felt, A.P., Reeder, R.W., Almuhiemedi, H., Consolvo, S.: Experimenting at scale with google chrome’s SSL warning. In: CHI (2014)
20. French, A.: Which? makes scams super-complaint-banks must protect those tricked into a bank transfer (2016), <https://www.which.co.uk/news/2016/09/which-makes-scams-super-complaint-453196/>
21. Garay, J.A., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: EURO-CRYPT (2015)
22. Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. In: CCS (2017)
23. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014), <https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269>
24. Kjørven, M.E.: Who pays when things go wrong? online financial fraud and consumer protection in scandinavia and europe. European Business Law Review (2020)
25. Küsters, R., Liedtke, J., Müller, J., Rausch, D., Vogt, A.: Ordinos: A verifiable tally-hiding e-voting system. In: EuroS&P (2020)
26. Laughery, K.R., Wogalter, M.S.: Designing effective warnings. Reviews of human factors and ergonomics (2006)
27. Lending Standards Board: Contingent reimbursement model code for authorised push payment scams (2021), <https://www.lendingstandardsboard.org.uk/wp-content/uploads/2021/04/CRM-Code-LSB-Final-April-2021.pdf>
28. McIlroy, D., Sethi-Smith, R.: Prospects for bankers’ liability for authorised push payment fraud. Butterworths Journal of International Banking and Financial Law (2021), <https://www.forumchambers.com/wp-content/uploads/2021/03/Article-2-Smith.1-1.pdf>
29. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Tech. rep. (2019)
30. Ortolani, P.: Self-enforcing online dispute resolution: lessons from bitcoin. Oxford Journal of Legal Studies (2016)
31. Ortolani, P.: The impact of blockchain technologies and smart contracts on dispute resolution: arbitration and court litigation at the crossroads. Uniform law review (2019)
32. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO (1991)
33. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments. Tech. rep. (2016), <https://lightning.network/lightning-network-paper.pdf>
34. Schneier, B.: Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition. Wiley (1996)
35. Taylor, J.L., Galica, T.: A new code to protect victims in the uk from authorised push payments fraud. Banking & Finance Law Review (2020)
36. The European Union Agency for Law Enforcement Cooperation: Take control of your digital life. don’t be a victim of cyber scams! (2021), <https://www.europol.europa.eu/activities-services/public-awareness-and-prevention-guides/take-control-of-your-digital-life-don%E2%80%99t-be-victim-of-cyber-scams>
37. The Federal Reserve: Fraud classifier (2020), <https://fedpaymentsimprovement.org/wp-content/uploads/fraudclassifier-industry-adoption-roadmap.pdf>
38. The Financial Conduct Authority: The payment services regulations (2017), <https://www.legislation.gov.uk/uksi/2017/752/contents/made>
39. The Financial Ombudsman Service: Lending standards board review of the contingent reimbursement model code for authorised push payment scams-financial ombudsman service response (2020), <https://www.financial-ombudsman.org.uk/files/289009/2020-10-02-LSB-CRM-Code-Review-Financial-Ombudsman-Service-Response.pdf>
40. The International Criminal Police Organization: Investment fraud via dating apps (2021), <https://www.interpol.int/en/News-and-Events/News/2021/Investment-fraud-via-dating-apps>
41. UK Finance: 2021 half year fraud update (2021), <https://www.ukfinance.org.uk/system/files/Half-year-fraud-update-2021-FINAL.pdf>
42. UK Finance: The definitive overview of payment industry fraud (2021), <https://www.ukfinance.org.uk/system/files/Fraud%20The%20Facts%202021-%20FINAL.pdf>
43. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper (2014)
44. Zee, S.V.D.: Shifting the blame? investigation of user compliance with digital payment regulations. In: Cybercrime in Context. Springer (2021)
45. Zuo, M.: Online “pig butchering” love scams have gone global after getting their start in china (2021), <https://www.scmp.com/news/people-culture/social-welfare/article/3150688/online-pig-butchering-love-scams-have-gone>

A Variant 1 Encoding-Decoding Protocol's Main Theorem and Proof

Theorem 1. *Let set $S = \{s_1, \dots, s_m\}$ be the union of two disjoint sets S' and S'' , where S' contains non-zero random values pick uniformly from a finite field \mathbb{F}_p , S'' contains zeros, $|S'| \geq c' = 1$, $|S''| \geq c'' = 0$, and pair (c', c'') is public information. Then, $r = \bigoplus_{i=1}^m s_i$ reveals nothing beyond the public information.*

Proof. Let s_1 and s , be two random values picked uniformly at random from \mathbb{F}_p . Let $\bar{s} = s_1 \oplus \underbrace{0 \oplus \dots \oplus 0}_{|S''|}$.

Since $\bar{s} = s_1$, two values \bar{s} and s have identical distribution. Thus, \bar{s} reveals nothing in this case. Next, let $\tilde{s} = \underbrace{s_1 \oplus s_2 \oplus \dots \oplus s_j}_{|S'|}$, where $s_i \in S'$. Since each s_i is a uniformly random value, the XOR of them is a

uniformly random value too. That means values \tilde{s} and s have identical distribution. Thus, \tilde{s} reveals nothing in this case as well. Also, it is not hard to see that the combination of the above two cases reveals nothing too, i.e., $\bar{s} \oplus \tilde{s}$ and s have identical distribution. \square

B Variant 2 Encoding-Decoding Protocol's Main Theorem and Proof

Theorem 2. *Let set $S = \{s_1, \dots, s_m\}$ be a set of random values picked uniformly from \mathbb{F}_p , where the cardinality of S is public information. Let \mathbf{BF} be a Bloom filter encoding all elements of S . Then, \mathbf{BF} reveals nothing about any element of S , beyond the public information, except with a negligible probability in the security parameter, i.e., with a probability at most $\frac{|S|}{2^\lambda}$.*

Proof. First, we consider the simplest case where only a single element of S is encoded in \mathbf{BF} . In this case, due to the pre-image resistance of the Bloom filter's hash functions and the fact that the set's element was picked uniformly at random from \mathbb{F}_p , the probability that \mathbf{BF} reveals anything about the original element is at most $\frac{1}{2^\lambda}$. Now, we move on to the case where all elements of S are encoded in \mathbf{BF} . In this case, the probability that \mathbf{BF} reveals anything about at least an element of the set is $\frac{|S|}{2^\lambda}$, due to the pre-image resistance of the hash functions, the fact that all elements were selected uniformly at random from the finite field, and the union bound. Nevertheless, when a \mathbf{BF} 's size is set appropriately to avoid false-positive without wasting storage, this reveals the number of elements encoded in it, which is public information. Thus, the only information \mathbf{BF} reveals is the public one. \square

C Further Discussion on the Verdict Encoding-decoding Protocol

Recall that each variant of our verdict encoding-decoding protocol is a voting mechanism. It lets a third party, \mathcal{I} , find out if threshold arbiters voted 1, while (i) generating unlinkable verdicts, (ii) not requiring arbiters to interact with each other for each customer, (iii) hiding the number of 0 or 1 verdicts from \mathcal{I} , and (iv) being efficient. Therefore, it is natural to ask:

Is there any e-voting protocol that can simultaneously satisfy all the above requirements?

The short answer is no. Recently, a provably secure e-voting protocol that can hide the number of 1 and 0 votes has been proposed by Kusters *et al.* [25]. Although this scheme can satisfy the above security requirements, it imposes a high computation cost, as it involves computationally expensive primitives such as zero-knowledge proofs, threshold public-key encryption scheme, and generic multi-party computation. In contrast, our verdict encoding-decoding protocols rely on much more lightweight operations such as XOR and hash function evaluations. We also highlight that our verdict encoding-decoding protocols are in a different setting than the one in which most of the e-voting protocols are. Because the former protocols are in the setting where there exists a small number of arbiters (or voters) which are trusted and can interact with each other once; whereas, the latter (e-voting) protocols are in a more generic setting where there is a large number of voters, some of which might be malicious, and they are not required to interact with each other.

Note that each variant of our verdict encoding-decoding protocol requires every arbiter to provide an encoded vote in order for \mathcal{I} to extract the final verdict. To let each variant terminate and \mathcal{I} find out the final verdict in the case where a set of arbiters do not provide their vote, we can integrate the following idea into each variant. We define a manager arbiter, say \mathcal{D}_n , which is always responsive and keeps track of missing votes. After the voting time elapses and \mathcal{D}_n realises a certain number of arbiters did not provide their encoded vote, it provides 0 votes on their behalf and masks them using the arbiters' masking values.