Payment with Dispute Resolution: A Protocol for Reimbursing Frauds Victims

Anonymous Author(s)

ABSTRACT

An "Authorised Push Payment" (APP) fraud refers to a 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, the implementation is lacking in transparency, 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 implement its most computationally-intensive subroutine and analyse its runtime. We also evaluate its asymptotic cost. 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.

CCS CONCEPTS

• **Security and privacy → Cryptography**.

KEYWORDS

Financial fraud, dispute resolution, transparency, accountability

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. The APP fraud has various variants, such as romance, investment, or invoice fraud [37]. The total amount of money lost to APP fraud is substantial. According to statistics collected from the UK banking industry by "UK Finance", in the first half of 2021, a total of £355 million was lost to APP frauds. Losses have increased by 71% compared to those reported in the same period in 2020 and now makes up almost half of banking fraud losses in the UK [36]. APP fraud is a *global* phenomenon. According to the FBI, victims of APP fraud reported to it at least a total of $419 million losses, in 2020 [17]. Recently, Interpol warned its member countries about a variant of APP fraud called investment fraud via dating software [35]. According to Europol's notice, at least five variants of APP fraud are among the seven most common types of online financial fraud [33].

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 [36]. Despite the UK's financial regulators (unlike US and EU) having provided specific reimbursement regulations to financial institutes to improve APP fraud victims' protection, these regulations are ambiguous and open to interpretation. Also, *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. It is not uniform among all banks and even among those organisations that settle disputes between banks and customers. 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, we propose a protocol called "Payment with Dispute Resolution" (PwDR), present its formal definition, and prove the protocol's security. The PwDR lets a victim (of an APP fraud) independently prove its innocence to a (potentially semi-honest) third-party neutral 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 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 can rely on and extend the security of the existing banking system. It automates the implementation of 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 *novel 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. The evaluation indicates that the protocol is efficient. The customer's and bank's complexity is constant, $O(1)$. It only takes 0.09 milliseconds for a dispute resolver to settle a dispute between the two parties.

2 BACKGROUND

Losses resulting from APP fraud fall outside legislation that protects customers from "unauthorised" payments (e.g., the Payment Services Directive 2 in the EU and the Electronic Fund Transfer Act in the US). 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 fraud rate is growing, victims do not have enough protection [18]. 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 [26] 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 for protecting the victims of fraud, some aspects of it are vague and open to interpretation. For instance, in 2020, the "Financial Ombudsman Service" (which 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 [34].

3 RELATED WORK

3.1 Authorised Push Payment Fraud

Anderson *et al.* [3] 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.* [32] analyse 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. Nevertheless, they also argue that the code is still ambiguous. Kjorven [23] 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 banks.

3.2 Dispute Resolution

In payment platforms, dispute resolution mechanisms can be broadly split into two classes, (a) *centralised* and (b) *decentralised*. In the former class, at any point in time, a *single party* tries to settle a 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, banks' terms and conditions (T&C) can complicate the dispute resolution process. If they do not reach an agreement, the customer can take its case to a third party (e.g., Financial Ombudsman Service or court) to settle the dispute. In 2000, Bohm *et al.* [9] 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 [2] points out that the move to online banking led many financial institutions to impose T&C that ultimately would shift the burden of proof in dispute to the customer. Becker *et al.* [7] investigate to what extent bank customers know the T&C they signed up for. Their study suggests that only 35% of customers fully understand T&C and 28% of customers find important parts of T&C are unclear.

Now we turn our attention to the latter class, i.e., decentralised dispute resolution. After the invention of 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 [10, 27, 28]. 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.* [14] propose FairSwap, an efficient protocol that lets a seller and buyer 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 [15] propose OPTISWAP that improves FairSwap's performance. Similar to FairSwap, OPTISWAP uses a Merkle tree and smart contract, but it uses an *interactive* dispute resolution protocol. Abadi *et al.* [1] propose a protocol that allows a fair exchange of digital coins and a certain *digital service*, called "proofs of data retrievability". To efficiently settle disputes between a seller and a buyer, the protocol uses blockchain, Merkle tree, and a third-party arbiter.

However, (the above) fair exchange schemes (including those that use blockchain and Merkle tree) cannot fully solve the problem that PwDR solves; because (1) PwDR captures the generic case where users use online banking and *fiat currency* to transfer money, whereas in the fair exchange protocols party *must have and trade in cryptocurrency*, and (2) PwDR captures various variants of APP fraud (e.g., CEO, romance, or invoice) where victims do not necessarily expect to receive an item in exchange for the money they transfer, whereas fair exchange protocols only allow parties want to exchange digital items, e.g., coins and files/services. Thus, PwDR offers a generic solution that fair exchange protocols cannot offer.

In the context of verifiable (cloud) computation, Dong *et al.* [13] use a combination of smart contracts, game theory, 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.* [20] propose a variant of payment channel [29] (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.

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

4 PRELIMINARIES

4.1 Taxonomy and Assumptions

A payment with a 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 6.

- Customer (C): it is a 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. We 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., registry office) which provides signed digital certificates (e.g., certificate of disability, divorce) to customers.
- A committee of auditors ($\mathcal{D}_1$, ..., $\mathcal{D}_n$): it consists of trusted third-party authorities or regulators (e.g., FCA, financial ombudsman service). They compile complaints and provide their verdicts. 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 secure under a Chosen-Plaintext Attack (CPA).
- 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 C and $\mathcal{B}$ use a secure channel when they send a message directly to $\mathcal{DR}$.

4.2 Digital Signature

A digital signature is a scheme for verifying the authenticity of digital messages and is formally defined in [22] as below.

Definition 1. A signature scheme involves three algorithms; namely, ($\mathtt{Sig.keyGen}$, $\mathtt{Sig.sign}$, $\mathtt{Sig.ver}$) that are defined as follows. (1) $\mathtt{Sig.keyGen}(1^\lambda) \rightarrow (sk, pk)$ is 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. (2) $\mathtt{Sig.sign}(sk, pk, u) \rightarrow sig$ is an algorithm run by the signer. It takes as input key pair: (sk, pk) and a message: u. It outputs a signature: sig. (3) $\mathtt{Sig.ver}(pk, u, sig) \rightarrow h \in \{0, 1\}$ is an 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 must meet two properties: (1) *Correctness:* for every input u it holds that: $Pr\Big[\ \mathtt{Sig.ver}(pk, u, \mathtt{Sig.sign}(sk, pk, u)) = 1 \ : \ \mathtt{Sig.keyGen}(1^\lambda) \rightarrow (sk, pk)\Big] = 1$. And (2) *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^*, except with a small probability, σ. Formally: $Pr\Big[u^* \notin Q \ \wedge \ \mathtt{Sig.ver}(pk, u^*, sig^*) = 1 \ : \ \mathtt{Sig.keyGen}(1^\lambda) \rightarrow (sk, pk), \mathcal{A}^{\mathtt{Sig.sign}(sk,)}(pk) \rightarrow (u^*, sig^*)\Big] \leq \mu(\lambda)$, where Q is the set of queries that $\mathcal{A}$ sent to the oracle.

4.3 Smart Contract

A smart contract is a computer program/code; it encodes the terms and conditions of an agreement between parties and often contains a set of variables and functions. A smart contract code is stored on a blockchain and is maintained by the miners who maintain the blockchain. When (a function of) a smart contract is triggered by an external party, every miner executes the smart contract's code. The program execution's correctness is guaranteed by the security of the underlying blockchain. Ethereum [38] has been the most predominant cryptocurrency framework that lets users define *arbitrary* smart contracts.

4.4 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 $\mathtt{Com}(x, r) = \mathtt{Com}_x$, that involves a secret value, r. In the open phase, the sender sends the opening $\ddot{x} := (x, r)$ to the receiver which verifies its correctness: $\mathtt{Ver}(\mathtt{Com}_x, \ddot{x}) \stackrel{?}{=} 1$ and accepts if the output is 1. A commitment scheme satisfies two properties, (a) *hiding*: it is infeasible for an adversary to learn any information about the message, and (b) *binding*: it is infeasible for an adversary to open a commitment to different values than the one used in the commit phase.

4.5 Statement Agreement Protocol

The "Statement Agreement Protocol" (SAP) proposed in [1] lets two mutually distrusted parties, e.g., $\mathcal{B}$ and C, efficiently agree on a private statement, π. The SAP satisfies 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. It assumes that each party has a blockchain public address, $adr_{\mathcal{R}}$ (where $\mathcal{R} \in \{\mathcal{B}, C\}$). Below, we restate the SAP.

(1) **Initiate**. $\mathtt{SAP.init}(1^\lambda, adr_{\mathcal{B}}, adr_C, \pi)$.
The following steps are taken by $\mathcal{B}$.
  (a) Deploys a smart contract that states both parties' addresses, $adr_{\mathcal{B}}$ and adr_C. Let adr_{SAP} be the deployed contract's address.
  (b) Picks a random value r, and commits to π as $\mathtt{Com}(\pi, r) = g_{\mathcal{B}}$. It sends adr_{SAP} and $\ddot{\pi} := (\pi, r)$ to C, and $g_{\mathcal{B}}$ to the contract.

(2) **Agreement**. $\mathtt{SAP.agree}(\pi, r, g_{\mathcal{B}}, adr_{\mathcal{B}}, adr_{\text{SAP}})$.
The following steps are taken by C.
  (a) Checks if $g_{\mathcal{B}}$ was sent from $adr_{\mathcal{B}}$, and checks locally $\mathtt{Ver}(g_{\mathcal{B}}, \ddot{\pi}) = 1$.
  (b) If the checks pass, it sets $b = 1$, computes locally $\mathtt{Com}(\pi, r) = g_C$, and sends g_C to the contract. Else, it sets $b = 0$ and $g_C = \perp$.

(3) **Prove**. For either $\mathcal{B}$ or C to prove, it sends $\ddot{\pi} := (\pi, r)$ to the smart contract.

(4) **Verify**. $\mathtt{SAP.verify}(\ddot{\pi}, g_{\mathcal{B}}, g_C, adr_{\mathcal{B}}, adr_C)$.
The following steps are taken by the smart contract.
  (a) Ensures $g_{\mathcal{B}}$ and g_C were sent from $adr_{\mathcal{B}}$ and adr_C respectively. It also ensures $\mathtt{Ver}(g_{\mathcal{B}}, \ddot{\pi}) = \mathtt{Ver}(g_C, \ddot{\pi}) = 1$.
  (b) Outputs $s = 1$, if the checks in steps 4a pass. It outputs $s = 0$, otherwise.

4.6 Pseudorandom Function

A pseudorandom function is a deterministic function that takes a key of length Λ and an input; it outputs a value indistinguishable from that of a truly random function [22]. In this paper, we use the pseudorandom function: $\mathtt{PRF} : \{0, 1\}^\Lambda \times \{0, 1\}^* \rightarrow \mathbb{F}_p$, where p is a large prime number, $|p| = \lambda$, and (Λ, λ) are the security parameters.

4.7 Bloom Filter

A Bloom filter [8] is a compact data structure that lets us efficiently check an element membership. It is an array of m bits (initially all set to zero), that represents n elements. It is accompanied by 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 membership, all its hash values are re-computed and checked whether all are set to 1 in the filter. If all the corresponding bits are 1, then the element is probably in the filter; otherwise, it is not. In this work, we require that a Bloom filter uses *cryptographic* hash functions. In the paper's full version [6], we explain how its parameters can be set.

5 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.

5.1 Challenge 1: Lack of Transparent Logs

In the current online banking system, during a payment journey, messages exchanged between a customer and a bank are 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.

5.2 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 a fraud 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 neutral 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 determine it.

5.3 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.

5.4 Challenge 4: Preserving Privacy

Although the use of a public logging mechanism is vital in resolving disputes transparently, if it does not use a privacy-preserving mechanism, parties' privacy would be violated. To protect the privacy of the bank's and customers' messages from 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 $\mathcal{DR}$ learns 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.

6 DEFINITION OF PAYMENT WITH DISPUTE RESOLUTION SCHEME

This section outlines a formal definition of the payment with dispute resolution (pwdr). We refer readers to the paper's full version [6] for a more detailed formal definition.

Definition 2. A pwdr involves six types of entities; namely, bank $\mathcal{B}$, customer C, smart contract $\mathcal{S}$, certificate generator $\mathcal{G}$, set of auditors $\mathcal{D} : \{\mathcal{D}_1, ..., \mathcal{D}_n\}$, and dispute resolver $\mathcal{DR}$. It also includes the following algorithms.

- $\texttt{keyGen}(1^\lambda) \rightarrow (sk, pk)$. It is run independently by $\mathcal{G}$ and one of the auditors, $\mathcal{D}_j$. It generates and outputs a pair of secret keys $sk := (sk_\mathcal{G}, sk_\mathcal{D})$ and public keys $pk := (pk_\mathcal{G}, pk_\mathcal{D})$, where $sk_\mathcal{D}$ may include multiple secret keys.
- $\texttt{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$. It is run by $\mathcal{B}$. It outputs an encoding-decoding token T (where $T := (T_1, T_2)$, each T_i contains a secret value $\ddot{\pi}_i$ and its public witness g_i), set of public parameters pp (including a threshold parameter e), and empty list $\boldsymbol{l}$.
- $\texttt{customerInit}(1^\lambda, T, pp) \rightarrow a$. It is run by C. It is an initiation algorithm that checks the correctness of the elements in T and pp. If the checks pass, it outputs 1. Else, it outputs 0.
- $\texttt{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$. It is run by C. It uses the new payee's detail f and encoding algorithm $\texttt{Encode}(T_1, .)$ to generate an encoded update request $\hat{m}_1^{(C)}$. It outputs $\hat{m}_1^{(C)}$.
- $\texttt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$. It is run by $\mathcal{S}$. It inserts a new payee's detail into $\boldsymbol{l}$ and outputs an updated list $\hat{\boldsymbol{l}}$.
- $\texttt{genWarning}(T, \hat{\boldsymbol{l}}, aux) \rightarrow \hat{m}_1^{(\mathcal{B})}$. It is run by $\mathcal{B}$. It outputs an encoded (warning) message $\hat{m}_1^{(\mathcal{B})}$, with the help of auxiliary data aux and $\texttt{Encode}(T_1, .)$, where the plaintext message is either "pass" or "warning" string.
- $\texttt{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$. It is run by C. It generates an encoded payment request $\hat{m}_2^{(C)}$, with the help of new payment's detail in_f and $\texttt{Encode}(T_1, .)$. It outputs $\hat{m}_2^{(C)}$.
- $\texttt{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$. It is run by $\mathcal{B}$. It generates and outputs an encoded message $\hat{m}_2^{(\mathcal{B})}$ for confirmation of payment.
- $\texttt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, aux_f) \rightarrow (\hat{z}, \hat{\ddot{\pi}})$. It is run by C. It generates complaints with the help of auxiliary data aux_f. If C 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 $u \in aux_f$, the evidence's

certificate $sig \in aux_f$, the certificate's public parameter, and "challenge warning", where the certificate is obtained from $\mathcal{G}$ via a query, Q. If its complaint is about the payment, it sets z_3 to "challenge payment". It generates and outputs (i) encoded complaints $\hat{z}$ using $\texttt{Encode}(T_1, .)$, and (ii) encoded secret parameters $\hat{\bar{\pi}}$ using another encoding algorithm $\bar{\texttt{Encode}}(pk_{\mathcal{D}}, .)$.

- $\texttt{verComplaint}(\hat{z}, \hat{\bar{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j$. It is run by every $\mathcal{D}_j$. It compiles j-th auditor's complaints. It initially sets parameters as $w_{1,j} = w_{2,j} = w_{3,j} = w_{4,j} = 0$. If the complaint in z_1 is valid, it sets $w_{1,j} = 1$. If the certificate in z_2 is valid, it sets $w_{3,j} = 1$. It checks the warning's effectiveness, by running $\texttt{checkWarning}(.)$. If it is not effective, i.e., $\texttt{checkWarning}(m_1^{(\mathcal{B})}) = 0$, it sets $w_{2,j} = 1$. Also, if the payment was indeed made, it sets $w_{4,j} = 1$. It outputs encoded verdicts $\hat{\boldsymbol{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]$ for j-th auditor.
- $\texttt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v}$. It is run by $\mathcal{DR}$. It aggregates all encoded verdicts $\hat{\boldsymbol{w}} = [\hat{\boldsymbol{w}}_1, ..., \hat{\boldsymbol{w}}_n]$ and outputs $\boldsymbol{v} = [v_1, ..., v_4]$, where $v_i = 1$ if at least e verdicts $w_{i,j}$ is 1; otherwise, $v_i = 0$. If $v_4 = 1$ and (i) either $v_1 = 1$ (ii) or $v_2 = 1$ and $v_3 = 1$, then C is reimbursed.

A pwdr has two properties, *correctness* and *security*. Correctness requires that the payment journey is completed (in the absence of a fraudster) without the need for (i) the honest customer to complain and (ii) the honest bank to reimburse. A pwdr is secure if it meets three main properties, (a) security against a malicious victim, (b) security against a malicious bank, and (c) privacy.

Informally, security against a malicious victim states that an APP fraud victim who is not qualified for the reimbursement should not be reimbursed. Specifically, a corrupt victim cannot (i) make at least the threshold of the auditors, $\mathcal{D}_j$s, conclude that $\mathcal{B}$ should have provided a warning, although $\mathcal{B}$ has done so, or (ii) 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 (iii) persuade at least the threshold of $\mathcal{D}_j$s to conclude that the warning was ineffective although it was effective, or (iv) make $\mathcal{DR}$ believe that the warning message was ineffective although only less than the threshold of $\mathcal{D}_j$s believe it, or (v) convince $\mathcal{D}_j$s to accept an invalid certificate, or (vi) make $\mathcal{DR}$ believe that at least the threshold of $\mathcal{D}_j$s accepted the certificate although they did not. Below, we formally state it.

Definition 3 (Security against a malicious victim). A pwdr is secure against a malicious victim, if for any security parameter λ, auxiliary data aux, and PPT adversary $\mathcal{A}$, there is a negligible function $\mu(\cdot)$, such that for experiment $\text{Exp}_1^{\mathcal{A}}$:

$\text{Exp}_1^{\mathcal{A}}(1^\lambda, aux)$

$\texttt{keyGen}(1^\lambda) \rightarrow (sk, pk)$
$\texttt{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$
$\mathcal{A}(1^\lambda, T, pp, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$
$\texttt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$
$\texttt{genWarning}(T, \hat{\boldsymbol{l}}, aux) \rightarrow \hat{m}_1^{(\mathcal{B})}$
$\mathcal{A}(T, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$
$\texttt{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$
$\mathcal{A}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk) \rightarrow (\hat{z}, \hat{\bar{\pi}})$
$\forall j, j \in [n]:$
$\left(\texttt{verComplaint}(\hat{z}, \hat{\bar{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j\right)$
$\texttt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v} = [v_1, ..., v_4]$

it holds that the following probability is negligible (i.e., $\mu(\lambda)$):

$$\Pr\left[\begin{array}{l}
\Big(\underbrace{(m_1^{(\mathcal{B})} = warning) \wedge (\sum_{j=1}^{n} w_{1,j} \geq e)}_{\text{(i)}}\Big) \vee \\
\Big(\underbrace{(\sum_{j=1}^{n} w_{1,j} < e) \wedge (v_1 = 1)}_{\text{(ii)}}\Big) \vee \\
\Big(\underbrace{(\texttt{checkWarning}(m_1^{(\mathcal{B})}) = 1) \wedge (\sum_{j=1}^{n} w_{2,j} \geq e)}_{\text{(iii)}}\Big) \vee \\
\Big(\underbrace{(\sum_{j=1}^{n} w_{2,j} < e) \wedge (v_2 = 1)}_{\text{(iv)}}\Big) \vee \\
\Big(\underbrace{u \notin Q \wedge \texttt{Sig.ver}(pk, u, sig) = 1}_{\text{(v)}}\Big) \vee \\
\Big(\underbrace{(\sum_{j=1}^{n} w_{3,j} < e) \wedge (v_3 = 1)}_{\text{(vi)}}\Big)
\end{array} : \text{Exp}_1^{\mathcal{A}}(\text{input})\right]$$

where $\hat{\boldsymbol{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]$, $\hat{\boldsymbol{w}} = [\hat{\boldsymbol{w}}_1, ..., \hat{\boldsymbol{w}}_n]$, $\hat{\boldsymbol{m}} = [\hat{m}_1^{(C)}, \hat{m}_2^{(C)}, \hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}]$, $(w_{1,j}, ..., w_{3,j})$ are the decoding of $(\hat{w}_{1,j}, ..., \hat{w}_{3,j}) \in \hat{\boldsymbol{w}}_j \in \hat{\boldsymbol{w}}$, and $\text{input} := (1^\lambda, aux)$.

Security against a malicious bank requires that a malicious bank cannot disqualify an honest victim from being reimbursed. Specifically, a corrupt bank cannot (i) make $\mathcal{DR}$ conclude that the "pass" message was correctly given or an important warning was not missing although at least the threshold of $\mathcal{D}_j$s do not believe so, or (ii) convince $\mathcal{DR}$ that the warning message was effective although at least the threshold of $\mathcal{D}_j$s do not believe so, or (iii) 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 it, or (iv) make $\mathcal{DR}$ believe that no payment was made, although at least the threshold of $\mathcal{D}_j$s believe the opposite.

Definition 4 (Security against a malicious bank). A pwdr scheme is secure against a malicious bank, if for any λ, aux, and PPT adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$, such that for an experiment $\text{Exp}_2^{\mathcal{A}}$:

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

$\texttt{keyGen}(1^\lambda) \rightarrow (sk, pk)$
$\mathcal{A}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l}, f, in_f, aux_f)$
$\texttt{customerInit}(1^\lambda, T, pp) \rightarrow a$
$\texttt{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$
$\texttt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$
$\mathcal{A}(T, \hat{\boldsymbol{l}}, aux) \rightarrow \hat{m}_1^{(\mathcal{B})}$
$\texttt{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$
$\mathcal{A}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$
$\texttt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, aux_f) \rightarrow (\hat{z}, \hat{\bar{\pi}})$
$\forall j, j \in [n]:$
$\left(\texttt{verComplaint}(\hat{z}, \hat{\bar{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j\right)$
$\texttt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v} = [v_1, ..., v_4]$

it holds that the following probability is $\mu(\lambda)$:

$$\Pr\left[\begin{array}{l} \Big(\underbrace{(\sum_{j=1}^{n} w_{1,j} \geq e) \wedge (v_1 = 0)}_{\text{(i)}}\Big) \vee \\ \Big(\underbrace{(\sum_{j=1}^{n} w_{2,j} \geq e) \wedge (v_2 = 0)}_{\text{(ii)}}\Big) \vee \\ \Big(\underbrace{(\sum_{j=1}^{n} w_{3,j} \geq e) \wedge (v_3 = 0)}_{\text{(iii)}}\Big) \vee \\ \Big(\underbrace{(\sum_{j=1}^{n} w_{4,j} \geq e) \wedge (v_4 = 0)}_{\text{(iv)}}\Big) \end{array} : \mathsf{Exp}_2^{\mathcal{A}}(\text{input})\right]$$

A pwdr scheme is privacy-preserving if it protects the privacy of (1) customers, bank, and auditors' sensitive messages from the scheme's non-participants and (2) each auditor's verdict from $\mathcal{DR}$.

Definition 5 (Privacy). A pwdr scheme preserves privacy if the following two properties are satisfied.

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

$\mathsf{Exp}_3^{\mathcal{A}_1}(1^\lambda, aux)$

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

it holds that:

$$\Pr\left[\mathcal{A}_1(g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}, \hat{\bar{\pi}}, \hat{\boldsymbol{w}}) \to \gamma : \mathsf{Exp}_3^{\mathcal{A}_1}(\text{input})\right] \leq \frac{1}{2} + \mu(\lambda)$$

(2) For any PPT 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 $\mathsf{Exp}_4^{\mathcal{A}_2}$:

$\mathsf{Exp}_4^{\mathcal{A}_2}(1^\lambda, aux)$

$\mathsf{keyGen}(1^\lambda) \to (sk, pk)$
$\mathsf{bankInit}(1^\lambda) \to (T, pp, \boldsymbol{l})$
$\mathsf{customerInit}(1^\lambda, T, pp) \to a$
$\mathcal{A}_2(1^\lambda, pk, a, pp, \boldsymbol{l}) \to (f, in_f, aux_f)$
$\mathsf{genUpdateRequest}(T, f, \boldsymbol{l}) \to \hat{m}_1^{(C)}$
$\mathsf{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \to \hat{\boldsymbol{l}}$
$\mathcal{A}_2(T, \hat{\boldsymbol{l}}, aux) \to m_1^{(\mathcal{B})}$
$\mathsf{Encode}(T_1, m_1^{(\mathcal{B})}) \to \hat{m}_1^{(\mathcal{B})}$
$\mathsf{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \to \hat{m}_2^{(C)}$
$\mathcal{A}_2(T, pk, aux_f, \hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(C)}) \to (m_2^{(\mathcal{B})}, z, \bar{\pi})$
$\mathsf{Encode}(T_1, m_2^{(\mathcal{B})}) \to \hat{m}_2^{(\mathcal{B})}$
$\mathsf{Encode}(T_1, z) \to \hat{z}$
$\mathsf{Encode}(pk_{\mathcal{D}}, \bar{\pi}) \to \hat{\bar{\pi}}$
$\forall j, j \in [n]$:
$\Big(\mathsf{verComplaint}(\hat{z}, \hat{\bar{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \to \hat{\boldsymbol{w}}_j\Big)$
$\mathsf{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \to \boldsymbol{v}$

it holds that:

$$\Pr\left[\begin{array}{l}\mathcal{A}_3(T_2, pk, pp, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}, \hat{\bar{\pi}}, \\ \hat{\boldsymbol{w}}, \boldsymbol{v}) \to w_j\end{array} : \mathsf{Exp}_4^{\mathcal{A}_2}(\text{input})\right] \leq Pr' + \mu(\lambda)$$

Let $\mathcal{D}_i$ output 0 and 1 with probabilities $Pr_{i,0}$ and $Pr_{i,1}$ respectively. Then, Pr' is defined as $Max(Pr_{1,0}, Pr_{1,1}, ..., Pr_{n,0}, Pr_{n,1})$.

Definition 6 (Security). A pwdr scheme is secure if it meets security against a malicious victim, security against a malicious bank, and preserves privacy with respect to definitions 3, 4, and 5 respectively.

We refer readers to Appendix A and the paper's full version [6] for further discussion about the above definitions.

7 PAYMENT WITH DISPUTE RESOLUTION PROTOCOL

In this section, we first present an outline of the PwDR (in Section 7.1). Then, we present a few subroutines (in Sections 7.2–7.4) that will be used in this protocol. After that, we describe the PwDR in detail (in Section 7.5).

7.1 An Overview of the Protocol

At a high level, the PwDR works as follows. 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. 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 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 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 the 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 C's complaint, then 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 C.

7.2 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.

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

- *Input.* $add_{\mathcal{S}}$: the address of smart contract $\mathcal{S}$, $m^{(\mathcal{B})}$: $\mathcal{B}$'s warning message, $\boldsymbol{l}$: 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 $m^{(\mathcal{B})}$ ="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 $\boldsymbol{l}$, with the help of aux.
  - if $\boldsymbol{l}$ contains an invalid element, it sets $w_1 = 1$.
  - otherwise, it sets $w_1 = 0$.
(3) returns w_1.

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

7.3 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.

7.4 Subroutines for Encoding-Decoding Verdicts

Now, 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

`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: 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 $z = m||sig||pk||$"challenge warning". If sig is empty, it sets $w_3 = 0$ and goes to step 2. Otherwise, it:
  (a) verifies the certificate: $\mathtt{Sig.ver}(pk, m, sig) \rightarrow h$.
  (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).

Figure 2: Algorithm to Check Warning's Effectiveness.

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 one is generic and works for any threshold (but is less efficient).

7.4.1 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. In Appendix B, we present the above observation's formal statement and its proof. At a high level, PVE and FVD work as follows. The auditors only once agree on a secret key (to do that one of them picks a random key and sends it to the rest). 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.[1] 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

[1] It is similar to the idea used in the XOR-based secret sharing [30].

the auditors voted 1, so the final verdict is 1. Figures 3 and 4 present PVE and FVD respectively.

7.4.2 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. Appendix D presents the above observation's formal statement and proof. 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, also derived from the above key. Thus, there would be a set $A = \{\alpha_1, ..., \alpha_n\}$ 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 $A = \{\alpha_1, ..., \alpha_n\}$. If multiple elements are taken at a time (i.e., $i > 1$), they are XORed with each other. Let $W = \{(\alpha_1 \oplus ... \oplus \alpha_e), (\alpha_2 \oplus ... \oplus \alpha_{e+1}), ..., (\alpha_1 \oplus ... \oplus \alpha_n)\}$ be the result. $\mathcal{D}_n$ computes each element of W regardless of what a specific auditor votes; also, it can generate each α_i independently, as it knows the single key (of the pseudorandom function) that was used by other auditors to generate these values. To protect the votes representations' privacy (from $\mathcal{I}$), it inserts all elements of W into a Bloom filter. Let BF be the resulting Bloom filter. It sends BF to $\mathcal{I}$.

In GFVD, to decode 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 $c \in \mathsf{BF}$. 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 ($c \notin \mathsf{BF}$), it learns that less than the threshold of the auditors voted 1; so, it sets the final verdict to 0. Figures 6 and 7, in Appendix C, present the GPVE and GFVD protocols in detail. Note, the total number of the combinations, i.e., W's cardinality, is small when the number of auditors is not high. In general, due to the binomial theorem, W's cardinality is determined as: $|W| = \sum_{i=e}^{n} \frac{n!}{i!(n-i)!}$. For instance, when $n = 10$ and $e = 6$, then $|W| = 386$. Appendix E provides more discussions on the above protocols and their difference from existing voting protocols.

7.5 The PwDR

In this section, we present the PwDR in detail.

(1) *Generating $\mathcal{G}$'s and $\mathcal{D}_j$'s Parameters*: $\mathsf{keyGen}(1^\lambda) \rightarrow (sk, pk)$.
Parties $\mathcal{G}$ and (only) $\mathcal{D}_j$ take steps 1a and 1b respectively.
   (a) calls $\mathsf{Sig.keyGen}(1^\lambda) \rightarrow (sk_{\mathcal{G}}, pk_{\mathcal{G}})$ to generate secret key $sk_{\mathcal{G}}$ and public key $pk_{\mathcal{G}}$. It publishes $pk_{\mathcal{G}}$.
   (b) calls $\tilde{\mathsf{keyGen}}(1^\lambda) \rightarrow (\tilde{sk}_{\mathcal{D}}, \tilde{pk}_{\mathcal{D}})$ to generate decrypting secret key $\tilde{sk}_{\mathcal{D}}$ and encrypting public key $\tilde{pk}_{\mathcal{D}}$. It also generates a key $\bar{k}_0$ for PRF, i.e., $\bar{k}_0 \xleftarrow{\$} \{0,1\}^\Lambda$. It sets $pk_{\mathcal{D}} = \tilde{pk}_{\mathcal{D}}$ and $sk_{\mathcal{D}} := (\tilde{sk}_{\mathcal{D}}, \bar{k}_0)$. It publishes $pk_{\mathcal{D}}$ 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 occurs only once for all customers.

(2) *Bank-side Initiation*: $\mathsf{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{I})$.
Bank $\mathcal{B}$ takes the following steps.
   (a) picks secret keys $\bar{k}_1$ and $\bar{k}_2$ for a symmetric key encryption scheme secure against a Chosen-Plaintext Attack (CPA). It sets two private statements as $\pi_1 = \bar{k}_1$ and $\pi_2 = \bar{k}_2$.
   (b) calls $\mathsf{SAP.init}(1^\lambda, adr_{\mathcal{B}}, adr_C, \pi_i) \rightarrow (r_i, g_i, adr_{\mathrm{SAP}})$ to initiate agreements on statements $\pi_i \in \{\pi_1, \pi_2\}$ with C. Let $T_i := (\ddot{\pi}_i, g_i)$ and $T := (T_1, T_2)$, where $\ddot{\pi}_i := (\pi_i, r_i)$ 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 C and sends public parameter $pp := (adr_{\mathrm{SAP}}, \Delta)$ to smart contract $\mathcal{S}$.

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

- *Input.* $\bar{k}_0$: a key of pseudorandom function $\mathsf{PRF}(.)$, ID: a unique identifier, w_j: a verdict, o: a counter, 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 $j < n : r_j = \mathsf{PRF}(\bar{k}_0, o||j||\mathrm{ID})$.
   - 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 $\bar{w}_j$.

Figure 3: Private Verdict Encoding (PVE) Protocol. In the figure, $\mathcal{D}_n$ can generate other auditors' r_i values, given $\bar{k}_0$. Note, ID is a unique identifier (e.g., wallet address) of the party for whom a verdict is provided (e.g., a client), and o is a counter that determines how many times a verdict for the same ID holder has been generated in the past. ID and o are used to ensure that each r_j will be different for each invocation of PVE although the same key $\bar{k}_0$ is used.

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

- *Input.* n: the total number of auditors, and $\bar{\boldsymbol{w}} = [\bar{w}_1, ..., \bar{w}_n]$: 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, $\bar{w}_j \in \bar{\boldsymbol{w}}$, 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.

Figure 4: Final Verdict Decoding (FVD) Protocol.

(3) *Customer-side Initiation*: $\texttt{customerInit}(1^\lambda, T, pp) \rightarrow a$.
Customer C takes the following steps.
(a) calls $\texttt{SAP.agree}(\pi_i, r_i, g_i, adr_{\mathcal{B}}, adr_{\text{SAP}}) \rightarrow (g_i', b_i)$, to locally 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 C are honest, then $g_i = g_i'$. It also checks Δ in $\mathcal{S}$, e.g., to see if it is sufficiently large.
(b) 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*: $\texttt{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$.
Customer C takes the following steps.
(a) 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 $\boldsymbol{l}$ that should change to f.

(b) at time t_0, sends to $\mathcal{S}$ the encryption of $m_1^{(C)}$, i.e., $\hat{m}_1^{(C)} = \texttt{Enc}(\bar{k}_1, m_1^{(C)})$.

(5) *Inserting New Payee*: $\texttt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$.
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 $\hat{\boldsymbol{l}}$, resulting in an updated list, $\hat{\boldsymbol{l}}$.
- if $\hat{m}_1^{(C)}$ is empty, it does nothing.

(6) *Generating Warning*: $\texttt{genWarning}(T, \hat{\boldsymbol{l}}, aux) \rightarrow \hat{m}_1^{(\mathcal{B})}$.
Bank $\mathcal{B}$ takes the following steps.
(a) checks if the most recent list $\hat{\boldsymbol{l}}$ is not empty. If it is empty, it halts. Else, it proceeds to the next step.
(b) decrypts each element of $\hat{\boldsymbol{l}}$ and checks its correctness, e.g., checks whether each element meets its internal policy stated in aux. If the check passes, it sets $m_1^{(\mathcal{B})}$ = "pass". Otherwise, it sets $m_1^{(\mathcal{B})}$ = warning, where the warning is a string that contains a warning's detail concatenated with the string "warning".
(c) at time t_1, sends to $\mathcal{S}$ the encryption of $m_1^{(\mathcal{B})}$, i.e., $\hat{m}_1^{(\mathcal{B})} = \texttt{Enc}(\bar{k}_1, m_1^{(\mathcal{B})})$.

(7) *Generating Payment Request*: $\texttt{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$.
Customer C takes the following steps.
(a) at time t_2, decrypts $\hat{\boldsymbol{l}}$ and $\hat{m}_1^{(\mathcal{B})}$. Depending on the warning, 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 $\hat{\boldsymbol{l}}$ and amount it wants to send.
(b) at time t_3, sends to $\mathcal{S}$ the encryption of $m_2^{(C)}$, i.e., $\hat{m}_2^{(C)} = \texttt{Enc}(\bar{k}_1, m_2^{(C)})$.

(8) *Making Payment*: $\texttt{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$.
Bank $\mathcal{B}$ takes the following steps.
(a) at time t_4, decrypts $\hat{m}_2^{(C)}$, i.e., $m_2^{(C)} = \texttt{Dec}(\bar{k}_1, \hat{m}_2^{(C)})$.
(b) 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, $\texttt{pay}(in_f)$. In this case, it sets $m_2^{(\mathcal{B})}$="paid". Otherwise (i.e., if $m_2^{(C)} = \phi$ or neither checks pass), it sets $m_2^{(\mathcal{B})} = \phi$. It sends to $\mathcal{S}$ the encryption of $m_2^{(\mathcal{B})}$, i.e., $\hat{m}_2^{(\mathcal{B})} = \texttt{Enc}(\bar{k}_1, m_2^{(\mathcal{B})})$.

(9) *Generating Complaint*: $\texttt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, aux_f) \rightarrow (\hat{z}, \hat{\ddot{\pi}})$.
Customer C takes the following steps.
(a) decrypts $\hat{m}_1^{(\mathcal{B})}$ and $\hat{m}_2^{(\mathcal{B})}$; this results in $m_1^{(\mathcal{B})}$ and $m_2^{(\mathcal{B})}$ 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 C wants to make one of the two below statements, it sets z_1 ="challenge message".
  (i) the pass message (in $m_1^{(\mathcal{B})}$) 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 C wants to challenge the effectiveness of the warning (in $m_1^{(\mathcal{B})}$), 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 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 $\hat{z} = \texttt{Enc}(\bar{k}_1, z)$ and $\hat{\ddot{\pi}} = \tilde{\texttt{Enc}}(pk_{\mathcal{D}}, \ddot{\pi})$ to $\mathcal{S}$.

(10) *Verifying Complaint*: $\texttt{verComplaint}(\hat{z}, \hat{\ddot{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j$.
Every $\mathcal{D}_j \in \{\mathcal{D}_1, ..., \mathcal{D}_n\}$ acts as follows.
(a) at time t_7, decrypts $\hat{\ddot{\pi}}$, i.e., $\ddot{\pi} = \tilde{\texttt{Dec}}(\tilde{sk}_{\mathcal{D}}, \hat{\ddot{\pi}})$, where $\tilde{sk}_{\mathcal{D}} \in sk_{\mathcal{D}}$.
(b) checks the validity of $(\ddot{\pi}_1, \ddot{\pi}_2)$ in $\ddot{\pi}$ by locally running the SAP's verification, i.e., $\texttt{SAP.verify}(.)$, for each $\ddot{\pi}_i$. It returns s. If $s = 0$, it halts. If $s = 1$ for both $\ddot{\pi}_1$ and $\ddot{\pi}_2$, it proceeds to the next step.
(c) decrypts $\hat{\boldsymbol{m}} = [\hat{m}_1^{(C)}, \hat{m}_2^{(C)}, \hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}]$, using $\texttt{Dec}(\bar{k}_1, .)$, where $\bar{k}_1 \in \ddot{\pi}_1$. Let $[m_1^{(C)}, m_2^{(C)}, m_1^{(\mathcal{B})}, m_2^{(\mathcal{B})}]$ be the result.
(d) checks whether 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 C in $\mathcal{S}$. Also, it checks whether C made a payment request, by checking if $m_2^{(C)}$ is non-empty and (its encryption) was registered by C in $\mathcal{S}$ at time t_3. If either check fails, it halts.
(e) decrypts $\hat{z}$ and $\hat{\boldsymbol{l}}$ using $\texttt{Dec}(\bar{k}_1, .)$, where $\bar{k}_1 \in \ddot{\pi}_1$. Let $z := (z_1, z_2, z_3)$ and $\boldsymbol{l}$ be the result.
(f) sets its verdicts according to $z := (z_1, z_2, z_3)$ as follows.
- if "challenge message" $\notin z_1$, it sets $w_{1,j} = 0$. Otherwise, it runs $\texttt{verStat}(add_{\mathcal{S}}, m_1^{(\mathcal{B})}, \boldsymbol{l}, \Delta, aux) \rightarrow w_{1,j}$, to determine if a warning (in $m_1^{(\mathcal{B})}$) 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 $\texttt{checkWarning}(add_{\mathcal{S}}, z_2, m_1^{(\mathcal{B})}, aux') \rightarrow (w_{2,j}, w_{3,j})$, to determine the effectiveness of the warning (in $m_1^{(\mathcal{B})}$).
- if "challenge payment" $\in z_3$, it checks if the payment was made. If it passes, it sets $w_{4,j} = 1$. If it failes, it sets $w_{4,j} = 0$. If "challenge payment" $\notin z_3$, it checks if "paid" $\in m_2^{(\mathcal{B})}$. If it passes, it sets $w_{4,j} = 1$. Else, it sets $w_{4,j} = 0$.

(g) encodes its verdicts $(w_{1,j}, w_{2,j}, w_{3,j}, w_{4,j})$:
(i) maintains a counter, o_{adr_C}, for each C. It sets its initial value to 0. It skips this step, if the counter has already been set.
(ii) calls $\texttt{PVE}(.)$ to encode each verdict. In particular, it performs as follows. $\forall i, 1 \leq i \leq 4$:

- calls $\mathsf{PVE}(\bar{k}_0, adr_C, w_{i,j}, o_{adr_C}, n, j) \rightarrow \bar{w}_{i,j}$.
- sets $o_{adr_C} = o_{adr_C} + 1$.

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

(iii) uses $\bar{k}_2 \in \ddot{\pi}_2$ to further encode/encrypt $\mathsf{PVE}(.)$'s outputs as follows. $\hat{\boldsymbol{w}}_j = \mathsf{Enc}(\bar{k}_2, \bar{\boldsymbol{w}}_j)$.

(h) at time t_8, sends to $\mathcal{S}$ the encrypted vector, $\hat{\boldsymbol{w}}_j$.

(11) *Resolving Dispute*: $\mathsf{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v}$.
Party $\mathcal{DR}$ takes the below steps at time t_9, if it is invoked by C or $\mathcal{S}$ which sends $\ddot{\pi}_2 \in T_2$ to it.

(a) checks $\ddot{\pi}_2$'s validity by locally running the SAP's verification, i.e., $\mathsf{SAP.verify}(.)$, that returns s. If $s = 0$, it halts.

(b) computes the final verdicts, as below.

(i) uses $\bar{k}_2 \in \ddot{\pi}_2$ to decrypt the auditors' encoded verdicts, as follows. $\forall j, 1 \leq j \leq n : \bar{\boldsymbol{w}}_j = \mathsf{Dec}(\bar{k}_2, \hat{\boldsymbol{w}}_j)$, where $\hat{\boldsymbol{w}}_j \in \hat{\boldsymbol{w}}$.

(ii) constructs four vectors, $[\boldsymbol{u}_1, ..., \boldsymbol{u}_4]$, and sets each vector $\boldsymbol{u}_i$ as follows. $\forall i, 1 \leq i \leq 4 : \boldsymbol{u}_i = [\bar{w}_{i,1}, ..., \bar{w}_{i,n}]$, where $\bar{w}_{i,j} \in \bar{\boldsymbol{w}}_j$.

(iii) calls $\mathsf{FVD}(.)$ to extract each final verdict, as follows. $\forall i, 1 \leq i \leq 4 :$ calls $\mathsf{FVD}(n, \boldsymbol{u}_i) \rightarrow v_i$.

(c) outputs $\boldsymbol{v} = [v_1, ..., v_4]$.

Customer 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. To state it formally, the following relation must hold:

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

In the above protocol, even C and $\mathcal{B}$ that know the decryption secret keys, $(\bar{k}_1, \bar{k}_2)$, cannot link a certain verdict to an auditor, because: (a) they do not know the masking random values used by auditors to mask each verdict and (b) the final verdicts $(v_1, ..., v_4)$ 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 efficient. It is easy to replace them with GPVE and GFVD.

We highlight that our protocol does not require the bank to commit any funds to the smart contract, which keeps it consistent with the traditional banking setting. The final (legal) verdict would suffice to enforce the bank to reimburse victims. Furthermore, in the real world, during a payment journey, a customer may receive various warning messages depending on the details it provides, its transaction history, and the checks a bank conducts, e.g., "Confirmation of Payee" [11]. Thus, we have included warning messages in our protocol to match the real-world banking setting. Appendix F provides more discussion about the deployment of PwDR.

Below, we present the security theorem of the PwDR.

Theorem 1. The above PwDR is secure, with regard to Definition 6, if the digital signature is existentially unforgeable under chosen message attacks, SAP, and the verdict encoding-decoding protocols (i.e., PVE and FVD) are secure, the symmetric key encryption and asymmetric key encryption are CPA-secure, the blockchain is immutable, and the correctness of PVE and FVD holds.

Proof Outline. Below, we present an overview of the proof. See Appendix G for detailed proof. PwDR is secure against a malicious C because a malicious C cannot: (1) persuade an auditor (and $\mathcal{DR}$) to accept a different (d)encryption key other than what was agreed between C and $\mathcal{B}$ in the initiation phase due to the binding property of the SAP's commitment, (2) come up with a valid signature/certificate on a message that has never been queried to the signing oracle due to the existential unforgeability of the digital signature scheme, and (3) frame an honest $\mathcal{B}$ for providing an invalid message, due to the immutability of the blockchain and the existential unforgeability of the digital signature.

PwDR is secure against a malicious $\mathcal{B}$ because (1) malicious $\mathcal{B}$ cannot persuade $\mathcal{DR}$ to accept a different decryption key due to the SAP's binding property, (2) the probability that multiple representations of verdict 1 cancel out each other is negligible due to the correctness of PVE-FVD, and (3) $\mathcal{B}$ cannot frame an honest C for providing an invalid message due to the immutability of blockchain and the existential unforgeability of the digital signature.

PwDR's privacy holds due to (1) the security of the symmetric and asymmetric key encryptions against CPA, (2) the SAP's hiding property, and (3) the privacy-preserving property of PVE-FVD. □

Note that since the smart contract in our PwDR merely acts as an immutable bulletin board, one may replace it with any other efficient tamper-evident logging mechanism, e.g., [12, 25, 31].

8 EVALUATION

In this section, we analyse the PwDR's complexities, its concrete runtime, and transaction latency. Tables 1 and 2 summarize the asymptotic and performance analysis respectively.

8.1 Computation Complexity

8.1.1 Cost of Customer C. In Phase 3, 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 asymmetric encryption once to encrypt the private statements' opening. Therefore, C's complexity is $O(1)$.

8.1.2 Cost of Bank $\mathcal{B}$. 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 $O(1)$.

8.1.3 Cost of Auditor $\mathcal{D}_j$. 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 C's and $\mathcal{B}$'s messages that were posted on $\mathcal{S}$ (this includes C's complaint). Recall, in the same phase, each auditor encodes its verdict using a verdict encoding protocol. Next, 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'

Table 1: 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 $\mathcal{D}_1, ..., \mathcal{D}_{n-1}$	✓	✓	$O(1)$	$O(1)$
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 $\mathcal{DR}$	✓	✓	$O(n)$	$O(1)$

Table 2: Parties' run-time (in ms) in verdict encoding-decoding protocols. n: the number of auditors and e: the threshold.

Party	$n=6$		$n=8$		$n=10$		$n=12$	
	$e=1$	$e=4$	$e=1$	$e=5$	$e=1$	$e=6$	$e=1$	$e=7$
Auditor $\mathcal{D}_n$	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

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 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 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 $O(1)$.

8.1.4 Cost of Dispute Resolver $\mathcal{DR}$. 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 $O(n)$ in the latter case (in which GFVD is invoked), with the 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 $O(n)$.

8.2 Communication Cost

Now, we analyse the communication cost of the PwDR. 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.

8.3 Concrete Performance Analysis

In this section, we study the protocol's 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 7.4). They were implemented in C++, see [4, 5] for the source code. To conduct the experiment, we used a MacBook Pro laptop with quad-core Intel Core *i*5, 2 GHz CPU, and 16 GB RAM. We ran the experiment on average 100 times. The prototype implementation uses the "Cryptopp" library[2] for cryptographic primitives, the "GMP" library[3] for arbitrary precision arithmetic, and the "Bloom Filter" library [4]. In the experiment, we set the false-positive rate in a Bloom filter to 2^{-40} and the finite field size to 128 bits. We used AES to implement PRF. Table 2 (in Section 8) provides the runtime of $\mathcal{D}_n$ and $\mathcal{DR}$ 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.

As Table 2 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 5 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 $\mathcal{DR}$.

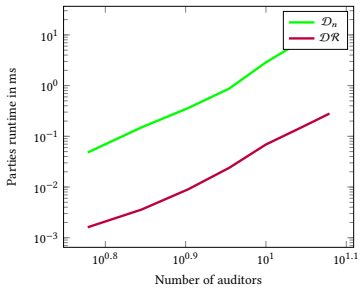

Figure 5: Parties' runtime in the PwDR.

[2] https://www.cryptopp.com
[3] https://gmplib.org
[4] http://www.partow.net/programming/bloomfilter/index.html

8.4 Transaction Latency

The transaction latency imposed by the blockchain to the PwDR depends on the type of consensus protocol used. For instance, on average it takes between 12 seconds in Ethereum to mine a block [16]; after this block is propagated to the network, to have adequate confidence that the block will remain in the chain, one may need to wait until at least 6 blocks are added after that block, which would take about 72 additional seconds. However, such a delay is lower in Byzantine Fault Tolerance (BFT) Hyperledger Fabric blockchain as it does not involve any mining and a consensus can be reached faster, e.g., about 35 seconds when 20 nodes are involved, [21].

9 CONCLUSION

In this work, to facilitate APP fraud 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. Our cost analysis indicated that PwDR is indeed efficient.

REFERENCES

[1] Aydin Abadi, Steven J. Murdoch, and Thomas Zacharias. 2021. Recurring Contingent Payment for Proofs of Retrievability. *IACR Cryptol. ePrint Arch.* (2021).

[2] Ross Anderson et al. 2007. Closing the phishing hole–fraud, risk and nonbanks. In *Federal Reserve Bank of Kansas City–Payment System Research Conferences*.

[3] Ross Anderson, Chris Barton, Rainer Bölme, Richard Clayton, Carlos Ganán, Tom Grasso, Michael Levi, Tyler Moore, and Marie Vasek. 2019. Measuring the changing cost of cybercrime. (2019).

[4] Anonymous. 2021. Variant 1: Efficient Verdict Encoding-Decoding Protocol. https://github.com/pwdrprotocol/PwDR/blob/main/encoding-decoding.cpp.

[5] Anonymous. 2021. Variant 2: Generic Verdict Encoding-Decoding Protocol. https://github.com/pwdrprotocol/PwDR/blob/main/generic-encoding-decoding.cpp.

[6] Anonymous. 2022. Payment with Dispute Resolution: A Protocol For Reimbursing Frauds Victims (Full Version). https://github.com/pwdrprotocol/PwDR/blob/main/PwDR.pdf.

[7] Ingolf Becker, Alice Hutchings, Ruba Abu-Salma, Ross J. Anderson, Nicholas Bohm, Steven J. Murdoch, M. Angela Sasse, and Gianluca Stringhini. 2017. International comparison of bank fraud reimbursement: customer perceptions and contractual terms. *J. Cybersecur.* (2017).

[8] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun.* (1970).

[9] Nicholas Bohm, Ian Brown, and Brian Gladman. 2000. Electronic Commerce: Who Carries the Risk of Fraud? *J. Inf. Law Technol.* 2000 (2000).

[10] Michael Buchwald. 2019. Smart contract dispute resolution: the inescapable flaws of blockchain-based arbitration. *U. Pa. L. Rev.* (2019).

[11] Confirmation of Payee Team. 2020. Confirmation of Payee- Response to consultation CP20/1 and decision on varying Specific Direction 10. (2020).

[12] Scott A. Crosby and Dan S. Wallach. 2009. Efficient Data Structures For Tamper-Evident Logging. In *USENIX Security*, Fabian Monrose (Ed.).

[13] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. 2017. Betrayal, Distrust, and Rationality: Smart Counter-Collusion Contracts for Verifiable Cloud Computing. In *CCS*.

[14] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. 2018. FairSwap: How To Fairly Exchange Digital Goods. In *CCS*.

[15] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. 2020. OptiSwap: Fast Optimistic Fair Exchange. In *ASIA CCS*.

[16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol. In *NSDI*.

[17] Federal Bureau of Investigation (FBI). 2020. Internet Crime Report. (2020). https://www.ic3.gov/Media/PDF/AnnualReport/2020_IC3Report.pdf.

[18] Adam French. 2016. Which? makes scams super-complaint-Banks must protect those tricked into a bank transfer. (2016).

[19] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT*.

[20] Matthew Green and Ian Miers. 2017. Bolt: Anonymous Payment Channels for Decentralized Currencies. In *CCS*.

[21] Hyperledger Foundation. 2018. Hyperledger Blockchain Performance Metrics.

[22] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, Second Edition.* CRC Press.

[23] Marte Eidsand Kjørven. 2020. Who pays when things go wrong? Online financial fraud and consumer protection in Scandinavia and Europe. *European Business Law Review* (2020).

[24] Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. 2020. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *EuroS&P*.

[25] Ben Laurie, Adam Langley, and Emilia Käsper. 2013. Certificate Transparency. *RFC* 6962 (2013), 1–27. https://doi.org/10.17487/RFC6962

[26] Lending Standards Board. 2021. 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.

[27] Pietro Ortolani. 2016. Self-enforcing online dispute resolution: lessons from bitcoin. *Oxford Journal of Legal Studies* (2016).

[28] Pietro Ortolani. 2019. The impact of blockchain technologies and smart contracts on dispute resolution: arbitration and court litigation at the crossroads. *Uniform law review* (2019).

[29] Joseph Poon and Thaddeus Dryja. 2016. *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.* Technical Report. https://lightning.network/lightning-network-paper.pdf.

[30] Bruce Schneier. 1996. *Applied cryptography - protocols, algorithms, and source code in C, 2nd Edition.* Wiley.

[31] Enrique Soriano-Salvador and Gorka Guardiola Muzquiz. 2021. SealFS: Storage-based tamper-evident logging. *Comput. Secur.* (2021).

[32] John L Taylor and Tony Galica. 2020. A New Code to Protect Victims in the UK from Authorised Push Payments Fraud. *Banking & Finance Law Review* (2020).

[33] The European Union Agency for Law Enforcement Cooperation. 2021. Take control of your digital life. Don't be a victim of cyber scams! 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.

[34] The Financial Ombudsman Service. 2020. 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.

[35] The International Criminal Police Organization. 2021. Investment fraud via dating apps. https://www.interpol.int/en/News-and-Events/News/2021/Investment-fraud-via-dating-apps.

[36] UK Finance. 2021. 2021 Half Year Fraud Update. https://www.ukfinance.org.uk/system/files/Half-year-fraud-update-2021-FINAL.pdf.

[37] UK Finance. 2021. THE DEFINITIVE OVERVIEW OF PAYMENT INDUSTRY FRAUD. https://www.ukfinance.org.uk/system/files/Fraud%20The%20Facts%202021-%20FINAL.pdf.

[38] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).

Appendix A MORE DISCUSSION ON DEFINITION 3

Definition 3 captures the case where `resDispute(.)` takes its inputs $\hat{w}$ and pp from a smart contract $\mathcal{S}$. Since there is a (negligible) probability that a smart contract's state can be tampered with, the definitions also capture such an event. Also, the definition aims to be as generic as possible, to capture the case where `resDispute(.)` can be either randomised/probabilistic or deterministic (in this paper, our PwDR that realises the definition relies on a deterministic dispute resolution algorithm). However, in a simpler case, where the parties send their messages directly to a fully trusted party (so the probability of tampering with its state 0) and/or `resDispute(.)` is deterministic, the definition can be slightly simplified.

Appendix B DISCUSSION ON PVE-FVD

First, we formally state our observation on which Variant 1 encoding-decoding protocol relies and then prove it. Then, we explain why this variant meets the three properties we laid out in Section 7.4, i.e., it should (1) generate unlinkable verdicts, (2) not require auditors to interact with each other for each customer, and (3) be efficient.

Theorem 2. Let set $S = \{s_1, ..., s_m\}$ 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 $\bar{s} = s_1 \oplus \underbrace{0 \oplus ... \oplus 0}_{|S''|}$. Since $\bar{s} = s_1$, two values $\bar{s}$ and s have identical distribution. Thus, $\bar{s}$ reveals nothing in this case. Next, let $\tilde{s} = s_1 \oplus s_2 \oplus ... \oplus s_j$, 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 distributions. 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. □

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 2). 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 $\frac{1}{2^\lambda}$. 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.

Appendix C GPVE & GFVD

In this section, we present the generic verdict encoding-decoding protocols (i.e., GPVE and GFVD) in detail, in Figures 6 and 7. They let a semi-honest third party $\mathcal{I}$ find out if at least e auditors voted 1, where e can be any integer in the range $[1, n]$. As Figure 6 indicates (and as we discussed in Section 7.4) after $\mathcal{D}_n$ generates all combinations of verdict 1's representations, it inserts the combinations into a Bloom filter, to preserve the representations' privacy from $\mathcal{I}$. Note, instead of inserting the combinations into a Bloom filter, we could *hash* the combinations and give the hash values to $\mathcal{I}$. However, using a Bloom filter lets us save considerable communication costs. Let us see a concrete example. Let $n = 10$, $e = 6$, and the hash function be SHA-256. If the latter (hash-based) approach is used, $\mathcal{D}_n$ needs to send $|W| \times 256 = 386 \times 256 = 98,816$ bits to $\mathcal{I}$, whereas if the former (Bloom filter-based) approach is used, then it only needs to send $|\mathsf{BF}| = 22,276$ bits to $\mathcal{I}$. Thus, by using a Bloom filter, it can save communication costs by at least a factor of 4.

Appendix D GPVE-GFVE'S PROOF

Below, we first formally state our main observation on which Variant 2 encoding-decoding protocol relies. After that, we prove it.

Theorem 3. Let $S = \{s_1, ..., s_m\}$ 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 $\frac{|S|}{2^\lambda}$.

Proof. First, we consider the simplest case where only a single element of S is encoded in 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 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

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

- *Input.* $\bar{k}_0$: a key of pseudorandom function $\mathsf{PRF}(.)$, ID: a unique identifier, w_j: a verdict, o: a counter, 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 $j < n : r_j = \mathsf{PRF}(\bar{k}_0, 1||o||j||\mathrm{ID})$.
   - 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.
$$w'_j = \begin{cases} 0, & \text{if } w_j = 0 \\ \alpha_j = \mathsf{PRF}(\bar{k}_0, 2||o||j||\mathrm{ID}), & \text{if } w_j = 1 \end{cases}$$

(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 $[e, n]$, computes the combinations (without repetition) of i elements from set $\{\alpha_1, ..., \alpha_n\}$. If multiple elements are taken at a time (i.e., $i > 1$), the elements are XORed with each other. Let $W = \{(\alpha_1 \oplus ... \oplus \alpha_e), (\alpha_2 \oplus ... \oplus \alpha_{e+1}), ..., (\alpha_1 \oplus ... \oplus \alpha_n)\}$ be the result.
   - constructs an empty Bloom filter. Then, it inserts all elements of W into this Bloom filter. Let BF be the Bloom filter encoding W's elements.

(5) outputs $(\bar{w}_j, \mathsf{BF})$.

Figure 6: Generic Private Verdict Encoding (GPVE) Protocol. In the figure, $\mathcal{D}_n$ can generate other auditors' r_i and α_j values, given $\bar{k}_0$. Note also that ID is a unique identifier (e.g., wallet address) of the party for whom a verdict is provided (e.g., a client), and o is a counter that determines how many times a verdict for the same ID holder has been generated in the past. ID and o are used to ensure each r_j will be different for each invocation of GPVE although the same key $\bar{k}_0$ is used.

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

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

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

(1) combines all auditors' encoded verdicts, $\bar{w}_j \in \bar{\boldsymbol{w}}$, 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,
$$v = \begin{cases} 0, & \text{if } c = 0 \text{ or } c \notin \mathsf{BF} \\ 1, & \text{if } c \in \mathsf{BF} \end{cases}$$

(4) outputs v.

Figure 7: Generic Final Verdict Decoding (GFVD) Protocol.

encoded in BF. In this case, the probability that 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 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. □

Appendix E (G)PVE-(G)FVD VERSUS EXISTING VOTING PROTOCOLS

Each variant of our verdict encoding-decoding protocol is a voting mechanism. It lets a third party, $\mathcal{I}$, find out if a threshold of the auditors voted 1, while (i) generating unlinkable verdicts, (ii) not requiring auditors 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, in the literature, 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 was proposed by Kusters *et al.* [24]. 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 auditors (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 auditor 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 auditors do not provide their vote, we can integrate the following idea into each variant. We define a manager auditor, 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 auditors did not provide their encoded vote, it provides 0 votes on their behalf and masks them using the auditors' masking values.

Appendix F THE DEPLOYMENT OF PWDR

The PwDR protocol is agnostic to the payment system it is enhancing. From a deployment perspective, the simplest solution is for the payment system provider to act on behalf of the customer in executing the smart contract. Of course, a malicious payment system provider could fail to register correct information in the smart contract but consumer organisations and regulators could check for such behaviour by inserting audit transactions and checking the blockchain. Alternatively, customers could make use of Payment Initiation Service Providers (PISPs) enabled by the Open Banking API of payment systems in the UK and EU. The PISP would be selected by the customer and act on behalf of the customer in both initiating the transfer with the bank and participating in the smart contract. A final option would be for the smart contract client to be integrated directly into the customer's online banking application either through integrating with documented APIs or by hooking functionality in the customer's web browser.

Appendix G SECURITY ANALYSIS OF PWDR

In this section, we prove the security theorem of the PwDR, i.e., Theorem 1. To prove it, we show that the PwDR satisfies all security properties defined in Section 6. 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 PwDR is secure against a malicious victim, w.r.t. Definition 3.

Proof. First, we focus on event I : $(m_1^{(\mathcal{B})} = warning) \wedge (\sum_{j=1}^{n} w_{1,j} \geq e)$ 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. Specifically, due to the security of SAP (i.e., the binding property of the SAP's commitment), $\mathcal{C}$ cannot convince an auditor to accept a different decryption key, e.g., $k' \in \ddot{\pi}'$, 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., $\bar{k}_1 \in \ddot{\pi}_1$. To be more precise, it cannot persuade an auditor to accept a statement $\ddot{\pi}'$, where $\ddot{\pi}' \neq \ddot{\pi}_1$, 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 : $(\sum_{j=1}^{n} w_{1,j} < e) \wedge (v_1 = 1)$ 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 binding property 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 $\ddot{\pi}'$, where $\ddot{\pi}' \neq \ddot{\pi}_2$, except with a negligible probability, $\mu(\lambda)$.

Now, we move on to event III : $(\texttt{checkWarning}(m_1^{(\mathcal{B})}) = 1) \wedge (\sum_{j=1}^{n} w_{2,j} \geq e)$. 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 : $(\sum_{j=1}^{n} w_{2,j} < e) \wedge (v_2 = 1)$ 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, 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 C and $\mathcal{B}$, except a negligible probability, $\mu(\lambda)$. Now, we analyse event V : $u \notin Q \wedge \texttt{Sig.ver}(pk, u, sig) = 1$. 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. But, 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 : $(\sum_{j=1}^{n} w_{3,j} < e) \wedge (v_3 = 1)$ that considers the case where less than the threshold of the auditors indicated that the signature (in 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 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 binding property, C cannot convince an auditor and $\mathcal{DR}$ to accept different encryption and decryption keys other than what was initially agreed between 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.

A malicious 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 [19]. □

Next, we present a lemma formally stating that the PwDR 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 PwDR is secure against a malicious bank, w.r.t. Definition 4.

Proof. We first focus on event I : $(\sum_{j=1}^{n} w_{1,j} \geq e) \wedge (v_1 = 0)$ 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, due to the SAP's binding property, $\mathcal{B}$ cannot persuade $\mathcal{DR}$ to accept a different decryption key, e.g., $k' \in \ddot{\pi}'$, other than what was agreed between C and $\mathcal{B}$ in the initiation phase, i.e., $\bar{k}_2 \in \ddot{\pi}_2$, except with a negligible probability. Specifically, it cannot persuade $\mathcal{DR}$ to accept a statement $\ddot{\pi}'$, where $\ddot{\pi}' \neq \ddot{\pi}_2$ except with probability $\mu(\lambda)$. Also, as discussed in Section 7.4, 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 : $(\sum_{j=1}^{n} w_{2,j} \geq e) \wedge (v_2 = 0)$, III : $(\sum_{j=1}^{n} w_{3,j} \geq e) \wedge (v_3 = 0)$, and IV : $(\sum_{j=1}^{n} w_{4,j} \geq e) \wedge (v_4 = 0)$ 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 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. □

Next, we prove the PwDR's privacy.

Lemma 3. If the symmetric key and asymmetric key encryption schemes are CPA-secure, and the SAP and encoding-decoding schemes (i.e., PVE and FVD) are secure, then PwDR is privacy-preserving, w.r.t. Definition 5.

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 $\hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}$, and $\hat{\boldsymbol{w}}$. Moreover, due to the semantical security of the symmetric key and asymmetric key encryption schemes, given ciphertext $(\hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}, \hat{\ddot{\pi}}, \hat{\boldsymbol{w}})$ 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 $\hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}$, and $\hat{z}$. Also, public parameters (pk, pp) and token T_2 are independent of 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 $(\hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}, \hat{\ddot{\pi}})$ 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 (i.e., Theorem 2), 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. Hence, excluding the case where all verdicts are 0, given $(T_2, pk, pp, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}, \hat{\ddot{\pi}}, \hat{\boldsymbol{w}}, \boldsymbol{v})$, $\mathcal{A}_3$'s success probability in experiment $\text{Exp}_4^{\mathcal{A}_2}$ to link a verdict to an auditor is at most $Pr' + \mu(\lambda)$. □

Theorem 4. The PwDR is secure, w.r.t. Definition 6.

Proof. Due to Lemma 1, the PwDR 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 6, meaning that the PwDR is indeed secure according to this definition. □