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

An "Authorised Push Payment" (APP) fraud refers to the case where fraudsters deceive a victim to make a payment to a bank account controlled by them. Total financial loss due to APP frauds is growing. Although regulators provided guidelines to improve victims' protection, the guidelines are vague and the victims are not receiving sufficient protection. To *facilitate victims' reimbursement*, in this work, we propose the notion of "Payment with Dispute Resolution" (PwDR) and formally define it. We also propose a candidate protocol and prove its security. The protocol lets an honest victim prove its innocence to a third-party dispute resolver (to be reimbursed) while preserving the involved parties' *privacy*. It makes black-box use of a standard online banking system. We evaluate its asymptotic cost and runtime via a prototype implementation. Our evaluation indicates that the protocol is efficient. It imposes only $O(1)$ overheads to the customer and bank. Also, it takes a dispute resolver only 0.09 milliseconds to settle a dispute between the two.

ACM Reference Format:
. 2021. Payment with Dispute Resolution: A Protocol For Reimbursing Frauds' Victims. In . ACM, New York, NY, USA, 29 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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, CEO, or invoice frauds [39]. The amount of money lost to APP frauds is substantial. According to a report produced by "UK Finance", only in the first half of 2021, a total of £355 million was lost to APP frauds, which has increased by 71% compared to losses reported in the same period in 2020 [38]. The UK Finance report suggests that online payment is the type of payment method the victims used to make the authorised push payment in 98% of cases. APP fraud is a *global* phenomenon. According to the FBI's report, victims of APP frauds reported to it at least a total of $419 million losses[1], in 2020 [18]. Recently, Interpol warned its member countries about a variant of APP fraud called investment fraud via dating software [37]. According to Europol's notice, at least five variants of APP fraud are among the seven most common types of online financial fraud [34].

Although the amount of money lost to 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 [38]. Despite the UK's financial regulators (unlike US and EU) have provided specific guidelines to financial institutes to improve APP frauds victims' protection, these guidelines are ambiguous and open to interpretation. Furthermore, there exists no mechanism in place via which honest victims can *prove* their innocence. To date, the APP fraud problem has been overlooked by the information security and cryptography research communities.

In this work, to facilitate the compensation of APP frauds' victims for their loss, we formally define a scheme called "Payment with Dispute Resolution" (PwDR), propose a protocol which instantiates it, and prove the protocol's security.

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

Manuscript submitted to ACM

The PwDR lets an honest victim (of an APP fraud) independently prove its innocence to a (potentially semi-honest) third-party dispute resolver, in order to be reimbursed. We identify three crucial properties that a PwDR scheme should possess; namely, (a) security against a malicious victim: a malicious victim which is not qualified for the reimbursement should not be reimbursed, (b) security against a malicious bank: a malicious bank cannot disqualify an honest victim from being reimbursed, and (c) privacy: the customer's and bank's messages remain confidential from non-participants of the scheme, and a party which resolves dispute learns as little information as possible. The protocol makes black-box use of a standard online banking system, meaning that it does not require significant changes to the existing online banking systems and can rely on their security. It is accompanied by our *lightweight privacy-preserving* threshold voting protocol, that can be of independent interest. We analyse the protocol's cost via both asymptotic and runtime evaluation (via a prototype implementation). The analysis indicates that the protocol is indeed efficient. The customer's and bank's complexity are constant, $O(1)$. It only takes 0.09 milliseconds for a dispute resolver to settle a dispute between the two parties. We make the implementation source code publicly available. We hope that our result lays the foundation for future solutions that will protect victims of this concerning type of fraud.

In summary, our contributions are three-fold, we: (1) put forward the notion of Payment with Dispute Resolution (PwDR), identify its core security properties, and formally define that, (2) propose an efficient candidate construction (that realises the PwDR's definition), and formally prove its security, and (3) analyse the construction's cost.

2 BACKGROUND

Liability for APP frauds has largely remained with victims that authorised the payment. In the UK, there have been efforts to protect the victims. In 2016, the UK's consumer protection organisation, called "Which?", submitted a super-complaint to the "Financial Conduct Authority" (FCA) and raised its concerns that despite the APP frauds victims' rate is growing, the victims do not have enough protection [19]. Since then, the FCA has been collaborating with financial institutes to develop several initiatives that could improve the response when they occur. As a result, the "Contingent Reimbursement Model" (CRM) code [25] was proposed. This code lays out a set of requirements and explains under which circumstances customers should be reimbursed by their financial institutes when they fall victim to an APP fraud. So far, there are at least nine firms, comprising nineteen brands (e.g., Barclays, HSBC, Lloyds) signed up to the code.

Although the CRM code is a vital guideline towards protecting the frauds' victims, it is still vague and open to interpretation. For instance, in 2020, the "Financial Ombudsman Service" (that settles complaints between consumers and businesses) highlighted 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 [36].

Unlike the UK that has already recognised APP frauds and developed regulations for that, the US's financial industry has not distinguished between APP fraud and other types of fraud when labeling their overall fraud losses [1]. Thus, there exist no publicly available accurate reports and regulations concerning APP frauds provided by the financial industry in the US yet.[2] To address the issue, in 2020, the Federal Reserve introduced the "FraudClassifier Model" which classifies payment frauds more granularly [35]. It is yet to be seen how regulations related to APP frauds will be developed and how the payment industry will implement these regulations. Similarly, in the EU, there exists no specific regulation developed to explicitly capture APP frauds and protect the victims [23, 26].

[2] Although the FBI publishes an annual internet crime report, e.g., [18], that includes some variants of APP fraud as well, this report defines APP fraud's variants very broadly that can cover unrelated frauds too. Also, it only captures the cases where victims directly reported APP frauds' occurrence to the FBI; therefore, it excludes the cases where the victims reported the APP frauds' occurrence to their banks.

3 PRELIMINARIES

3.1 Notations

We summarise our notations in Table 1.

Table 1. Notation Table.

Symbol	Description
Enc(.)	Encryption algorithm of symmetric key encryption
Dec(.)	Decryption algorithm of symmetric key encryption
$\tilde{\text{Enc}}$(.)	Encryption algorithm of asymmetric key encryption
$\tilde{\text{Dec}}$(.)	Decryption algorithm of asymmetric key encryption
keyGen(.)	Key generator algorithm of asymmetric key encryption
Sig.keyGen(.)	Key generator algorithm of digital signature scheme
verStat(.)	Algorithm to determine $\mathcal{B}$'s message status
checkWarning(.)	Algorithm to check a warning's effectiveness
pay(.)	$\mathcal{B}$'s internal algorithm to transfers money
Com(.)	Commitment's commit
Ver(.)	Commitment's verify
H(.)	Hash function
PRF(.)	Pseudorandom function
$\mathcal{C}$	Customer
$\mathcal{B}$	Bank
$\mathcal{D}_1, ..., \mathcal{D}_n$	Arbiters
$\mathcal{DR}$	Dispute resolver
$\mathcal{S}$	Smart contract
$\mathcal{G}$	Certificate generator
SAP	Statement agreement protocol
PVE	Private verdict encoding protocol
FVD	Final verdict decoding protocol
GPVE	Generic private verdict encoding protocol
GFVD	Generic final verdict decoding protocol
$\oplus$	XOR operator
Δ	time parameter
add_I	Address of I
z_1	$\mathcal{C}$'s complaint about $\mathcal{B}$'s message status
z_2	$\mathcal{C}$'s complaint about a warning's effectiveness
z_3	$\mathcal{C}$'s complaint about payment message inconsistency
aux, aux'	Auxiliary information
PPT	Probabilistic polynomial time

Symbol	Description
$\boldsymbol{l}$	$\mathcal{C}$'s payees list
$\hat{\boldsymbol{l}}$	$\mathcal{C}$'s encoded payees list
k_0	A secret key of PRF
f	New payee's detail
in_f	Payment detail
$\hat{a}$	Encoded a
$\hat{m}_1^{(\mathcal{C})}$	$\mathcal{C}$'s encoded update request
$\hat{m}_2^{(\mathcal{C})}$	$\mathcal{C}$'s encoded payment request
$\hat{m}_1^{(\mathcal{B})}$	$\mathcal{B}$'s encoded warning message
$\hat{m}_2^{(\mathcal{B})}$	$\mathcal{B}$'s encoded payment message
sk and pk	Secret and public keys
$sk_{\mathcal{D}}$	Arbiters' secret key
pp	Public parameter
j	Arbiter's index, $1 \leq j \leq n$
T, T_1, T_2	Tokens, where $T := (T_1, T_2)$
w_1	Output of verStat(.)
(w_2, w_3)	Output of checkWarning(.)
$\bar{w}_j$	Output of PVE(.)
v	Output of FVD(.)
e	Threshold
$w_{i,j}$	Arbiter's plain verdict
o	Offset
r_j	Pseudorandom value
λ	Security parameter
π	Private statement
in_p	The input of pay(.)
ϕ	Null
n	Total number of arbiters
z	$\mathcal{C}$'s complaint, where $z := (z_1, z_2, z_3)$
$g := (g_1, g_2)$	Commitment values
BF	Bloom filter

3.2 Informal Thread Model and Assumptions

The PwDR scheme consists of six types of parties. Below, we informally explain each type of party's role and the security assumption we make about each of them. We will provide a formal definition of the PwDR scheme in Section 5.

- Customer $\mathcal{C}$: it is a regular customer of a bank. We call a customer a victim after it falls victim to an APP fraud. We assume a victim is corrupted by a non-colluding active (or malicious) adversary.
- Bank $\mathcal{B}$: it is a regular bank that provides a standard online banking system. We assume it is corrupted by a non-colluding active adversary. We assume any change to the source code of the online banking system is transparent and can be detected.
- Smart contract $\mathcal{S}$: it is a standard smart contract of a public blockchain (e.g., Ethereum). It mainly acts as a tamper-proof public bulletin board to store different parties' messages.
- Certificate generator $\mathcal{G}$: it is a trusted third party (e.g., hospital, registry office) which provides signed digital certificates (e.g., certificate of disability, divorce) to customers.
- A committee of arbiters $\{\mathcal{D}_1, ..., \mathcal{D}_n\}$: it consists of trusted third-party authorities or regulators (e.g., financial conduct authority, financial ombudsman service). They compile complaints and provide their verdicts. If needed, they can access the banking backend software to conduct investigations. We assume the arbiters interacted with each other to agree on (a) a secret key, $\bar{k}_0$, and (b) a pair of keys $(pk_{\mathcal{D}}, sk_{\mathcal{D}})$ of an asymmetric key encryption.

- Dispute resolver $\mathcal{DR}$: it is an aggregator of arbiters' 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 $\mathcal{DR}$.

3.3 Digital Signature

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

Definition 3.1. A signature scheme involves three algorithms, $\texttt{Signature} := (\texttt{Sig.keyGen}, \texttt{Sig.sign}, \texttt{Sig.ver})$, that are defined as follows. (1) $\texttt{Sig.keyGen}(1^\lambda) \rightarrow (sk, pk)$ is probabilistic algorithm run by a signer. It takes as input a security parameter. It outputs a key pair: (sk, pk), consisting of secret: sk, and public: pk keys. (2) $\texttt{Sig.sign}(sk, pk, u) \rightarrow sig$ 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) $\texttt{Sig.ver}(pk, u, sig) \rightarrow h \in \{0, 1\}$ is deterministic algorithm run by a verifier. It takes as input public key: pk, message: u, and signature: sig. It checks the signature's validity. If the verification passes, then it outputs 1; otherwise, it outputs 0.

A digital signature scheme should meet two properties. (1) *Correctness:* for every input u it holds that: $Pr\Big[\ \texttt{Sig.ver}(pk, u, \texttt{Sig.sign}(sk, pk, u)) = 1 \ : \ \texttt{Sig.keyGen}(1^\lambda) \rightarrow (sk, pk)\Big] = 1$. 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. More formally:

$$\Pr\Big[\ u^* \notin Q \ \wedge \ \texttt{Sig.ver}(pk, u^*, sig^*) = 1 \ \ : \ \ \texttt{Cer.keyGen}(1^\lambda) \rightarrow (sk, pk), \ \ \mathcal{A}^{\texttt{Sig.sign}(k,)}(pk) \rightarrow (u^*, sig^*) \ \Big] \leq \mu(\lambda)$$

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

3.4 Smart Contract

Cryptocurrencies, such as Bitcoin [27] and Ethereum [40], beyond offering a decentralised currency, support computations on transactions. In this setting, a certain computation logic is encoded in a computer program, called a *"smart contract"*. Although Bitcoin, the first decentralised cryptocurrency, supports smart contracts, the functionality of Bitcoin's smart contracts is limited. To address this limitation, Ethereum, as a generic smart contract platform, was designed. Thus far, Ethereum has been the most predominant cryptocurrency framework that lets users define arbitrary smart contracts. In this framework, a contract's code and its related data are held by every node in the blockchain's network. The program execution's correctness is guaranteed by the security of the underlying blockchain components. To prevent a denial of service attack, the framework requires a transaction creator to pay a fee, called *"gas"*.

3.5 Commitment Scheme

A commitment scheme involves two parties, *sender* and *receiver*, and includes two phases: *commit* and *open*. In the commit phase, the sender commits to a message x as $\texttt{Com}(x, r) = \texttt{Com}_x$, 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: $\texttt{Ver}(\texttt{Com}_x, \ddot{x}) \stackrel{?}{=} 1$ and accepts if the output is 1. Informally, a commitment scheme must satisfy two properties, (a) *hiding*: it is infeasible for an adversary to learn any information about the committed 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. We provide more detail about the commitment scheme in Appendix A.

3.6 Statement Agreement Protocol

The "Statement Agreement Protocol" (SAP) proposed in [2] lets two mutually distrusted parties, e.g., $\mathcal{B}$ and C, efficiently agree on a private statement, π. Informally, 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. The SAP uses a smart contract and commitment scheme. It assumes that each party has a blockchain public address, $adr_{\mathcal{R}}$ (where $\mathcal{R} \in \{\mathcal{B}, C\}$). Below, we restate the SAP, taken from [2].

(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 the statement, $\mathtt{Com}(\pi, r) = g_{\mathcal{B}}$.
(c) 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.
(b) Ensures $\mathtt{Ver}(g_{\mathcal{B}}, \ddot{\pi}) = \mathtt{Ver}(g_C, \ddot{\pi}) = 1$.
(c) Outputs $s = 1$, if the checks, in steps 4a and 4b, pass. It outputs $s = 0$, otherwise.

3.7 Pseudorandom Function

Informally, a pseudorandom function is a deterministic function that takes a key of length Λ and an input; it outputs a value indistinguishable from that of a truly random function. 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. In practice, a pseudorandom function can be obtained from an efficient block cipher. We refer readers to [22] for a formal definition of a pseudorandom function.

3.8 Bloom Filter

A Bloom filter [11] is a compact data structure that allows us to 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 Bloom filters, it is possible that an element is not in the set, but the membership query indicates it is, i.e., false positives. In this work, we ensure that the false positive probability is negligible, e.g., 2^{-40}. Also, we require that a Bloom filter uses *cryptographic* hash functions. In Appendix B, we explain how the Bloom filter's parameters can be set.

4 CHALLENGES TO OVERCOME

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

4.1 Challenge 1: Lack of Transparent Logs

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

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

One of the determining factors in the process of allocating liability to customers (after an APP fraud occurs) is paying attention to and following "warning(s)", according to the CRM code. However, there exists no publicly available study on the effectiveness of every

warning provided by a bank. Therefore, we cannot hold a customer accountable for becoming the fraud's victim, even if the related warnings are ignored. To address this challenge, we let a warning's effectiveness be determined on a case-by-case basis after an APP fraud takes place. The protocol lets a victim challenge a certain warning whose effectiveness will be assessed by a *committee*, i.e., a set of arbiters. In this setting, each member of the committee provides (an encoding of) its verdict to the smart contract, from which a dispute resolver retrieves all verdicts to find out the final verdict. The scheme ensures that the final verdict is in the customer's favor if at least threshold committee members 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 arbiters determines it (in a secure manner).

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

Recall that an APP fraud occurs when a payment is made. In the case where a bank sends (to the smart contract) a confirmation of payment message, it is not possible to automatically validate such a claim, as the money transfer occurs outside of the blockchain network. To address this challenge, the protocol lets a customer raise a dispute and report it to the smart contract when it detects an inconsistency. In this case, the above committee members investigate and provide their verdicts to the smart contract that allows the dispute resolver to extract the final verdict.

4.4 Challenge 4: Preserving Privacy

Although the use of a public transparent logging mechanism plays a vital role in resolving disputes, if it does not use a privacy-preserving mechanism, then parties' privacy would be violated. To protect the privacy of the bank's and customers' messages against the public (and other customers), the PwDR protocol lets the customer and bank provably agree on encoding-decoding tokens that let them encode their outgoing messages. Later, either party can provide the token to a third party which can independently check the token's correctness, and decode the messages. To protect the privacy of the committee members' verdicts from the dispute resolver, the PwDR ensures that the dispute resolver can learn only the final verdict without being able to link a verdict to a specific member of the committee or even learn the number of yes/1 and no/0 votes. To this end, we develop and use a novel threshold voting protocol.

5 DEFINITION OF PAYMENT WITH DISPUTE RESOLUTION SCHEME

In this section, we outline a formal definition of a PwDR scheme. We refer readers to Appendix C for a full version of the definition.

Definition 5.1. A PwDR scheme involves six types of entities; namely, bank $\mathcal{B}$, customer C, smart contract $\mathcal{S}$, certificate generator $\mathcal{G}$, set of arbiters $\mathcal{D} : \{\mathcal{D}_1, ..., \mathcal{D}_n\}$, and dispute resolver $\mathcal{DR}$. It also includes the following algorithms.

- $\mathtt{keyGen}(1^\lambda) \rightarrow (sk, pk)$. It outputs a pair of secret keys $sk := (sk_{\mathcal{G}}, sk_{\mathcal{D}})$ and public keys $pk := (pk_{\mathcal{G}}, pk_{\mathcal{D}})$.
- $\mathtt{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$. It is an initiation algorithm that 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 $\boldsymbol{l}$.
- $\mathtt{customerInit}(1^\lambda, T, pp) \rightarrow a$. It is an initiation algorithm. It checks the correctness of the elements in T and pp. If the checks pass, it outputs 1. Otherwise, it outputs 0.
- $\mathtt{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$. It is an update request algorithm. It uses the new payee's detail f and encoding algorithm $\mathtt{Encode}(T_1, .)$ to generate an encoded update request $\hat{m}_1^{(C)}$. It outputs $\hat{m}_1^{(C)}$.
- $\mathtt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$. It is an algorithm that inserts a new payee's detail into $\boldsymbol{l}$ and outputs an updated list $\hat{\boldsymbol{l}}$.
- $\mathtt{genWarning}(T, \hat{\boldsymbol{l}}, \text{aux}) \rightarrow \hat{m}_1^{(\mathcal{B})}$. It is a warning generating algorithm that outputs an encoded (warning) message $\hat{m}_1^{(\mathcal{B})}$, with the assistance of auxiliary information aux and $\mathtt{Encode}(T_1, .)$, where the decoded message is either pass or warning string.
- $\mathtt{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$. It an algorithm that generates an encoded payment request $\hat{m}_2^{(C)}$, with the help of new payment's detail in_f and $\mathtt{Encode}(T_1, .)$. It outputs $\hat{m}_2^{(C)}$.
- $\mathtt{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$. It generates and outputs an encoded message $\hat{m}_2^{(\mathcal{B})}$ for confirmation of payment.
- $\mathtt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, \text{aux}_f) \rightarrow (\hat{z}, \hat{\ddot{\pi}})$. It generates complaints with the help of auxiliary data aux_f. If it wants to complain that (i) pass message should have been a warning or (ii) no message was provided, it sets z_1 to "challenge message". If its complaint is about the warning's effectiveness, it sets z_2 to a combination of an evidence $u \in aux_f$, 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 (a) encoded complaints $\hat{z}$ using $\mathsf{Encode}(T_1, .)$, and (b) encoded secret parameters $\hat{\pi}$ using another encoding algorithm $\mathsf{Encode}(pk_{\mathcal{D}}, .)$.

- $\mathsf{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j$. It compiles j-th arbiter's complaints. It initially sets parameters $w_{1,j} = w_{2,j} = w_{3,j} = w_{4,j} = 0$. 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 algorithm $\mathsf{checkWarning}(.)$. If it is not effective, i.e., $\mathsf{checkWarning}(m_1^{(\mathcal{B})}) = 0$, 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 arbiter.
- $\mathsf{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v}$. It aggregates all arbiters' 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 arbiters' original 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 $v_3 = 1$, C is reimbursed.

Informally, a PwDR scheme has two properties; namely, *correctness* and *security*. Correctness requires that (in the absence of a fraudster) the payment journey is completed without the need for (i) the honest customer to complain and (ii) the honest bank to reimburse the customer. A PwDR scheme is secure if it satisfies three main properties; namely, (a) security against a malicious victim, (b) security against a malicious bank, and (c) privacy. Below, we present a brief formal definition of them. Intuitively, security against a malicious victim requires that the victim of an APP fraud which is not qualified for the reimbursement should not be reimbursed. More specifically, a corrupt victim cannot (a) make at least threshold committee members, $\mathcal{D}_j$s, conclude that $\mathcal{B}$ should have provided a warning, although $\mathcal{B}$ has done so, or (b) make $\mathcal{DR}$ conclude that the pass message was incorrectly given or a vital warning message was missing despite only less than threshold $\mathcal{D}_j$s believe so, or (c) persuade at least threshold $\mathcal{D}_j$s to conclude that the warning was ineffective although it was effective, or (d) make $\mathcal{DR}$ believe that the warning message was ineffective although only less than threshold $\mathcal{D}_j$s believe that, or (e) convince $\mathcal{D}_j$s to accept an invalid certificate, or (f) make $\mathcal{DR}$ believe that at least threshold $\mathcal{D}_j$s accepted the certificate although they did not, except for a negligible probability.

Definition 5.2 (Security against a malicious victim). A PwDR scheme is secure against a malicious victim, if for any security parameter λ, auxiliary information aux, and PPT adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$, such that for an experiment $\mathsf{Exp}_1^{\mathcal{A}}$:

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

$\mathsf{keyGen}(1^\lambda) \rightarrow (sk, pk)$, $\mathsf{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$, $\mathcal{A}(1^\lambda, T, pp, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$,
$\mathsf{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$, $\mathsf{genWarning}(T, \hat{\boldsymbol{l}}, \mathrm{aux}) \rightarrow \hat{m}_1^{(\mathcal{B})}$, $\mathcal{A}(T, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$,
$\mathsf{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$, $\mathcal{A}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk) \rightarrow (\hat{z}, \hat{\pi})$,
$\forall j, j \in [n] : \Big(\mathsf{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]\Big)$,
$\mathsf{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v} = [v_1, ..., v_4]$

it holds that:

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

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

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

Definition 5.3 (Security against a malicious bank). A PwDR scheme is secure against a malicious bank, if for any security parameter λ, auxiliary information aux, and PPT adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$, such that for an experiment $\mathsf{Exp}_2^{\mathcal{A}}$:

, ,

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

$\mathtt{keyGen}(1^\lambda) \to (sk, pk)$, $\mathcal{A}(1^\lambda) \to (T, pp, \boldsymbol{l}, f, in_f, \mathrm{aux}_f)$, $\mathtt{customerInit}(1^\lambda, T, pp) \to a$,
$\mathtt{genUpdateRequest}(T, f, \boldsymbol{l}) \to \hat{m}_1^{(C)}$, $\mathtt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \to \hat{\boldsymbol{l}}$, $\mathcal{A}(T, \hat{\boldsymbol{l}}, \mathrm{aux}) \to \hat{m}_1^{(\mathcal{B})}$,
$\mathtt{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \to \hat{m}_2^{(C)}$, $\mathcal{A}(T, \hat{m}_2^{(C)}) \to \hat{m}_2^{(\mathcal{B})}$,
$\mathtt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, \mathrm{aux}_f) \to (\hat{z}, \hat{\bar{\pi}})$,
$\forall j, j \in [n] : \Big(\mathtt{verComplaint}(\hat{z}, \hat{\bar{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \to \hat{\boldsymbol{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]\Big)$,
$\mathtt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \to \boldsymbol{v} = [v_1, ..., v_4]$

it holds that:

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

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

Informally, a PwDR scheme is privacy-preserving if it protects the privacy of (1) customers', bank's, and arbiters' sensitive messages from non-participants of the protocol, including other customers, and (2) each arbiter's verdict from $\mathcal{DR}$ which sees the final verdict.

Definition 5.4 (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 $\mathsf{Exp}_3^{\mathcal{A}_1}$:

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

$\mathtt{keyGen}(1^\lambda) \to (sk, pk)$, $\mathtt{bankInit}(1^\lambda) \to (T, pp, \boldsymbol{l})$, $\mathtt{customerInit}(1^\lambda, T, pp) \to a$,
$\mathcal{A}_1(1^\lambda, pk, a, pp, g, \boldsymbol{l}) \to \Big((f_0, f_1), (in_{f_0}, in_{f_1}), (\mathrm{aux}_{f0}, \mathrm{aux}_{f1})\Big)$,
$\gamma \xleftarrow{\$} \{0, 1\}$, $\mathtt{genUpdateRequest}(T, f_\gamma, \boldsymbol{l}) \to \hat{m}_1^{(C)}$, $\mathtt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \to \hat{\boldsymbol{l}}$,
$\mathtt{genWarning}(T, \hat{\boldsymbol{l}}, \mathrm{aux}) \to \hat{m}_1^{(\mathcal{B})}$, $\mathtt{genPaymentRequest}(T, in_{f_\gamma}, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \to \hat{m}_2^{(C)}$,
$\mathtt{makePayment}(T, \hat{m}_2^{(C)}) \to \hat{m}_2^{(\mathcal{B})}$, $\mathtt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, \mathrm{aux}_{f_\gamma}) \to (\hat{z}, \hat{\bar{\pi}})$,
$\forall j, j \in [n] : \Big(\mathtt{verComplaint}(\hat{z}, \hat{\bar{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \to \hat{\boldsymbol{w}}_j\Big)$, $\mathtt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \to \boldsymbol{v}$

it holds that:

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

(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 $\mathsf{Exp}_4^{\mathcal{A}_2}$:

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

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

it holds that:

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

where $\hat{\boldsymbol{m}} = [\hat{m}_1^{(C)}, \hat{m}_2^{(C)}, \hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}]$, $\hat{\boldsymbol{w}} = [\hat{\boldsymbol{w}}_1, .., \hat{\boldsymbol{w}}_n]$, and input $:= (1^\lambda, aux)$. Let arbiter $\mathcal{D}_i$ 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 5.5 (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 5.2, 5.3, and 5.4 respectively.

6 PAYMENT WITH DISPUTE RESOLUTION PROTOCOL

6.1 An Overview of the PwDR Protocol

In this section, we provide an overview of the PwDR protocol. Initially, only once $\mathcal{C}$ and $\mathcal{B}$ agree on a smart contract $\mathcal{S}$. They also use the SAP to provably agree on two private statements that include 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 system. It generates an update request that specifies the new payee's detail, encrypts the request, and sends the result to $\mathcal{S}$. After that, $\mathcal{B}$ decrypts and checks the request's validity, e.g., whether it meets its internal policy. Depending on the request's content, $\mathcal{B}$ generates a pass or warning message. It encrypts the message and sends the result to $\mathcal{S}$. Then, $\mathcal{C}$ checks $\mathcal{B}$'s message, and depending on its content, decides whether to make payment. If it decides to do so, then it sends an encrypted payment detail to $\mathcal{S}$. Next, $\mathcal{B}$ 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. In particular, it generates an encrypted complaint that could challenge the effectiveness of the warning and/or any payment inconsistency. It can include in the complaint an evidence/certificate, e.g., asserting that it falls into the vulnerable customer category as defined in the CRM code. $\mathcal{C}$ encrypts the complaint and sends to $\mathcal{S}$ the result and a proof asserting the secret key's correctness. Then, each committee member verifies the proof. If the verification passes, it decrypts and compiles $\mathcal{C}$'s complaint to generate a (set of) verdict. Each committee member encodes its verdict and sends the encoded verdict's encryption to $\mathcal{S}$. To resolve a dispute between $\mathcal{C}$ and $\mathcal{B}$, either of them can invoke $\mathcal{DR}$. To do so, they directly send to it one of the above secret keys and a proof asserting that key was generated correctly. $\mathcal{DR}$ verifies the proof. If the verification passes, it locally decrypts the encrypted encoded verdicts (retrieved from $\mathcal{S}$) and then combines the result to find out the final verdict. If the final verdict indicates the legitimacy of $\mathcal{C}$'s complaint, then $\mathcal{C}$ must be reimbursed. Note, the verdicts are encoded in a way that even after decrypting them, $\mathcal{DR}$ cannot link a verdict to a committee member or even figure out how many 1 or 0 verdicts were provided (except when all verdicts are 0). However, it can find out whether at least threshold committee members voted in favor of $\mathcal{C}$. Shortly, we present novel verdict encoding-decoding (voting) protocols that offer the above features.

6.2 A Subroutines for Determining Bank's Message Status

As we stated earlier, 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 committee members to deal with such complaints deterministically, we propose `verStat(.)` algorithm, which is run locally by each committee member. This algorithm is presented in Figure 1.

6.3 A Subroutine for Checking a Warning's Effectiveness

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

6.4 Subroutines for Encoding-Decoding Verdicts

In this section, we present verdict encoding and decoding protocols. They let a third party $\mathcal{I}$, e.g., $\mathcal{DR}$, learn if threshold arbiters voted 1, while satisfying the following requirements. The protocols should (1) generate unlinkable verdicts, (2) not require arbiters to interact with each other for each customer, and (3) be efficient. Since the second and third requirements are self-explanatory, we only explain the first one. Informally, the first property requires that the protocols should generate encoded verdicts and final verdict in a way that $\mathcal{I}$, given these values, cannot (a) link a verdict to an arbiter (except when all verdicts are 0), and (b) learn the total number of

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

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

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

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

- *Input.* $add_{\mathcal{S}}$: the address of smart contract $\mathcal{S}$, z: $\mathcal{C}$'s complaint, $m^{(\mathcal{B})}$: $\mathcal{B}$'s warning message, and aux': auxiliary information, e.g., guideline on warnings' effectiveness.
- *Output.* $w_2 = 0$: if the given warning message is effective; $w_2 = 1$: if the warning message is ineffective. Also, $w_3 = 1$: if the certificate in z is valid or no certificate is provided; $w_3 = 0$: if the certificate is invalid.

(1) parse $z = m||sig||pk||$"challenge warning". If sig is empty, it sets $w_3 = 0$ and goes to step 2. Otherwise, it:
   (a) verifies the certificate: Sig.ver(pk, m, sig) $\rightarrow h$.
   (b) if the certificate is rejected (i.e., $h = 0$), it sets $w_3 = 0$. It goes to step 4.
   (c) otherwise (i.e., $h = 1$), it sets $w_3 = 1$ and moves onto the next step.
(2) checks if "warning" $\in m^{(\mathcal{B})}$. If the check is passed, it proceeds to the next step. Otherwise, it aborts.
(3) checks the warning's effectiveness, with the assistance of the evidence m and auxiliary information aux'.
   - if it is effective, it sets $w_2 = 0$. Otherwise, it sets $w_2 = 1$.
(4) returns (w_2, w_3).

Fig. 2. Algorithm to Check Warning's Effectiveness

1 or 0 verdicts when they provide different verdicts. Shortly, we present two variants of verdict encoding and decoding protocol. The first variant is highly efficient and suitable when the threshold is 1. The second variant is generic and works for any threshold.

6.4.1 Variant 1: Efficient Verdict Encoding-Decoding Protocol. This variant includes two protocols, Private Verdict Encoding (PVE) and Final Verdict Decoding (FVD). They let $\mathcal{I}$ learn if at least one arbiter 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. At a high level, they work as follows. The arbiters only once agree on a secret key. This key will let each of them, in PVE, generate a pseudorandom masking value such that if all masking values are XORed, they would cancel out each other.[3] In PVE, each arbiter encodes its verdict by (i) representing it as a parameter 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 arbiters voted 0; so, the final verdict is 0. But, if the result is not 0, it knows that at least one of the arbiters voted 1, so the final verdict is 1. Figures 3 and 4 present PVE and FVD respectively. In Appendix D, we present the above observation's formal statement, its proof, and discussion on why this variant meets the three requirements.

6.4.2 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 arbiters 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 F presents the above observation's formal statement and proof.

[3]This is similar to the idea used in the XOR-based secret sharing [32].

In this variant also, the arbiters initially agree on a secret key used to generate a pseudorandom masking value. Each arbiter 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 arbiter masks its verdict representation by its masking value. It sends the result to $\mathcal{I}$. Also, arbiter $\mathcal{D}_n$ generates a set W of all combinations of arbiters' 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. Note, $\mathcal{D}_n$ computes these values regardless of what a specific arbiter votes. Let $W = \{(\alpha_1 \oplus ... \oplus \alpha_e), (\alpha_2 \oplus ... \oplus \alpha_{e+1}), ..., (\alpha_1 \oplus ... \oplus \alpha_n)\}$ be the result. To protect the privacy of the votes' representations (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 and extract the final verdict, $\mathcal{I}$ XORs all masked verdict representations which removes the masking values and XORs the representations. Let c be the result. If $c = 0$, $\mathcal{I}$ concludes that all arbiters voted 0; so, it sets the final verdict to 0. If $c \neq 0$, it checks if $c \in \mathsf{BF}$. If it is, then it concludes that at least threshold arbiters voted 1, so it sets the final verdict to 1. Otherwise ($c \notin \mathsf{BF}$), it learns that less than threshold arbiters voted 1; so, it sets the final verdict to 0. Figures 5 and 6, in Appendix E, present the GPVE and GFVD protocols in detail. The total number of the combinations, i.e., the cardinality of W, is small when the number of arbiters is not very high. In general, due to the binomial theorem, the cardinality of W is determined as follows: $|W| = \sum_{i=e}^{n} \frac{n!}{i!(n-i)!}$. For instance, when $n = 10$ and $e = 6$, then $|W|$ is only 386. Appendix G provides further discussion on the above protocols.

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

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

Arbiter $\mathcal{D}_j$ 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 $\bar{w}_j$.

Fig. 3. Private Verdict Encoding (PVE) Protocol

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

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

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

(1) combines all arbiters' 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.

Fig. 4. Final Verdict Decoding (FVD) Protocol

6.5 The PwDR Protocol

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

(1) *Generating $\mathcal{G}$'s and $\mathcal{D}_j$'s Parameters*: $\mathsf{keyGen}(1^\lambda) \rightarrow (sk, pk)$. Party $\mathcal{G}$ and an arbiter $\mathcal{D}_j$ 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 the public key, $pk_{\mathcal{G}}$.
   (b) calls $\tilde{\mathsf{keyGen}}(1^\lambda) \rightarrow (sk_{\mathcal{D}}, pk_{\mathcal{D}})$ to generate decrypting secret key $sk_{\mathcal{D}}$ and encrypting public key $pk_{\mathcal{D}}$. It publishes the public key $pk_{\mathcal{D}}$ and sends $sk_{\mathcal{D}}$ to the rest of the arbiters.

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

(2) *Bank-side Initiation*: $\mathsf{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$. Bank $\mathcal{B}$ takes the following steps.
   (a) picks secret keys $\bar{k}_1$ and $\bar{k}_2$ for symmetric key encryption scheme and pseudorandom function PRF respectively. It sets two private statements as $\pi_1 = \bar{k}_1$ and $\pi_2 = \bar{k}_2$.
   (b) calls $\mathsf{SAP.init}(1^\lambda, adr_{\mathcal{B}}, adr_C, \pi_i) \rightarrow (r_i, g_i, adr_{\text{SAP}})$ to initiate agreements on statements $\pi_i \in \{\pi_1, \pi_2\}$ with customer 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 $pp := (adr_{\text{SAP}}, \Delta)$ to smart contract $\mathcal{S}$.

(3) *Customer-side Initiation*: $\mathsf{customerInit}(1^\lambda, T, pp) \rightarrow a$. Customer C takes the following steps.
   (a) calls $\mathsf{SAP.agree}(\pi_i, r_i, g_i, adr_{\mathcal{B}}, adr_{\text{SAP}}) \rightarrow (g'_i, b_i)$, to check the correctness of parameters in $T_i \in T$ and (if accepted) to agree on these parameters, where $(\pi_i, r_i) \in \ddot{\pi}_i \in T_i$ and $1 \leq i \leq 2$. Note, if both $\mathcal{B}$ and C are honest, then $g_i = g'_i$. It also checks Δ in $\mathcal{S}$, e.g., to see whether it is sufficiently large.
   (b) if the above checks fail, it sets $a = 0$. Otherwise, it sets $a = 1$. It sends a to $\mathcal{S}$.

(4) *Generating Update Request*: $\mathsf{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$. 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., $\hat{m}_1^{(C)} = \mathsf{Enc}(\bar{k}_1, m_1^{(C)})$.

(5) *Inserting New Payee*: $\mathsf{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$. 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*: $\mathsf{genWarning}(T, \hat{\boldsymbol{l}}, aux) \rightarrow \hat{m}_1^{(\mathcal{B})}$. 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. Otherwise, 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., $\hat{m}_1^{(\mathcal{B})} = \mathsf{Enc}(\bar{k}_1, m_1^{(\mathcal{B})})$.

(7) *Generating Payment Request*: $\mathsf{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$. Customer C takes the following steps.
   (a) at time t_2, decrypts the content of $\hat{\boldsymbol{l}}$ and $\hat{m}_1^{(\mathcal{B})}$. Depending on the warning's content, it sets a payment request $m_2^{(C)}$ to ϕ or in_f, where in_f contains the payment's detail, e.g., the payee's detail in $\hat{\boldsymbol{l}}$ and the amount it wants to transfer.
   (b) at time t_3, sends to $\mathcal{S}$ the encryption of $m_2^{(C)}$, i.e., $\hat{m}_2^{(C)} = \mathsf{Enc}(\bar{k}_1, m_2^{(C)})$.

(8) *Making Payment*: $\mathsf{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$. Bank $\mathcal{B}$ takes the following steps.
   (a) at time t_4, decrypts $\hat{m}_2^{(C)}$, i.e., $m_2^{(C)} = \mathsf{Dec}(\bar{k}_1, \hat{m}_2^{(C)})$.
   (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, $\mathsf{pay}(in_f)$. 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., $\hat{m}_2^{(\mathcal{B})} = \mathsf{Enc}(\bar{k}_1, m_2^{(\mathcal{B})})$.

(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 (i.e., pass or warning) 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 the certificate generator $\mathcal{G}$), and $pk_{\mathcal{G}} \in pk$.
- if C wants to complain about the inconsistency of the payment (in $m_2^{(\mathcal{B})}$), it sets z_3 = "challenge payment"; else, it sets $z_3 = \phi$.

(b) at time t_6, sends to $\mathcal{S}$ the following values:

- the encryption of complaint z, i.e., $\hat{z} = \texttt{Enc}(\bar{k}_1, z)$.
- the encryption of $\ddot{\pi} := (\ddot{\pi}_1, \ddot{\pi}_2)$, i.e., $\hat{\ddot{\pi}} = \tilde{\texttt{Enc}}(pk_{\mathcal{D}}, \ddot{\pi})$.

(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 $\mathcal{D}_j \in \{\mathcal{D}_1, ..., \mathcal{D}_n\}$ takes the following steps.

(a) at time t_7, decrypts $\hat{\ddot{\pi}}$, i.e., $\ddot{\pi} = \tilde{\texttt{Dec}}(sk_{\mathcal{D}}, \hat{\ddot{\pi}})$.

(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 the content of $z := (z_1, z_2, z_3)$, as follows.

- if "challenge message" $\notin z_1$, it sets $w_{1,j} = 0$. Otherwise, it runs $\texttt{verStat}(add_{\mathcal{S}}, m_1^{(\mathcal{B})}, \boldsymbol{l}, \Delta, aux) \rightarrow w_{1,j}$, 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 the check passes, it sets $w_{4,j} = 1$. If the check fails, it sets $w_{4,j} = 0$. If "challenge payment" $\notin z_3$, it checks if "paid" $\in m_2^{(C)}$. If it passes, it sets $w_{4,j} = 1$. Otherwise, it sets $w_{4,j} = 0$.

(g) encodes its verdicts $(w_{1,j}, w_{2,j}, w_{3,j}, w_{4,j})$ as follows.

(i) locally maintains a counter, o_{adr_C}, for each C. It sets its initial value to 0.

(ii) calls $\texttt{PVE}(.)$ to encode each verdict. In particular, it performs as follows. $\forall i, 1 \leq i \leq 4$:

- calls $\texttt{PVE}(\bar{k}_0, adr_C, w_{i,j}, o_{adr_C}, n, j) \rightarrow \bar{w}_{i,j}$.
- 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 $\texttt{PVE}(.)$'s outputs as follows. $\hat{\boldsymbol{w}}_j = \texttt{Enc}(\bar{k}_2, \bar{\boldsymbol{w}}_j)$.

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

(11) *Resolving Dispute*: $\texttt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v}$. 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 the validity of $\ddot{\pi}_2$ by locally running the SAP's verification, i.e., $\texttt{SAP.verify}(.)$, 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 arbiters' encoded verdicts, as follows. $\forall j, 1 \leq j \leq n$: $\bar{\boldsymbol{w}}_j = \texttt{Dec}(\bar{k}_2, \hat{\boldsymbol{w}}_j)$, 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 $\texttt{FVD}(.)$ to extract each final verdict, as follows. $\forall i, 1 \leq i \leq 4$: calls $\texttt{FVD}(n, \boldsymbol{u}_i) \rightarrow v_i$.

(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. Stated formally, the following relation must hold: $\Big(\underbrace{(v_1 = 1)}_{\text{(i)}} \vee \underbrace{(v_2 = 1 \ \wedge \ v_3 = 1)}_{\text{(ii)}}\Big) \wedge \Big(\underbrace{v_4 = 1}_{\text{(iii)}}\Big)$. Note, in the above PwDR 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 arbiter, for two main reasons; namely, (a) they do not know the masking random values used by arbiters 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 arbiters vote 0. We highlight that we used PVE and FVD in the PwDR protocol only because they are highly efficient. However, it is easy to replace them with GPVE and GFVD, e.g., when the required threshold is greater than one. We present the PwDR protocol's main theorem and its proof in Appendix H.

7 EVALUATION

We analysed the PwDR protocol's asymptotic cost. Table 2 summarises the analysis result. As the table indicates, the customer's and bank's complexity is constant. However, one of the arbiters, i.e., arbiter $\mathcal{D}_n$, and the dispute resolver have non-constant complexities which are mainly imposed by our verdict inducing-decoding protocols. Therefore, to study these parties' runtime in the PwDR, we implemented both variants of the verdict encoding-decoding protocols. The source code of variants 1 and 2 is available in [7] and [8] respectively. Table 3 summarises the study's result. As this table shows, the overall cost is low. In particular, 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. Appendix I provides more details about the above analysis.

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

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

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

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

8 RELATED WORK

8.1 Authorised Push Payment Fraud

Anderson *et al.* [6] provide an overview of APP frauds and highlight that although the (CRM) code would urge banks to accept more liability for APP frauds, it remains to be seen how this will evolve as fraudsters will continuously try to figure out how bank systems can facilitate misdirection attacks. Taylor *et al.* [33] analyses the CRM code from legal and practical perspectives. They state that this code's proper implementation would make considerable advances to protect victims of APP frauds. However, 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.

8.2 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 the dispute. In particular, if a customer disputes having made or authorised a transaction, then the related bank tries directly settle the dispute with the customer. However, the banks' terms and conditions can complicate the dispute resolution process. If they do not reach an agreement, the client can take its case to a third party (e.g., Financial Ombudsman Service or court) to settle the dispute. In 2000, Bohm *et al.* [12] 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 [5] points out that the move to online banking led many financial institutions to impose terms and conditions on their customers that ultimately would shift the burden of proof in dispute to the customer. Becker *et al.* [10] investigate to what extent bank customers know the terms and conditions (T&C) they signed up to. Their study suggests that only 35% of customers fully understand T&C and 28% of customers find important parts of T&C are unclear. 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 [14, 28, 29]. Moreover, various ad-hoc blockchain-based cryptographic protocols have been proposed to resolve disputes in various contexts and settings, e.g., in [2, 16, 17, 21, 31].

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

8.3 Central Bank Digital Currency

Now, we briefly discuss a different but related area, "Central Bank Digital Currency" (CBDC). Due to the growing interest in digital payments, some central banks around the world have been exploring the idea of CBDC [4]. The idea behind CBDC is that a central bank issues digital money/tokens to the public where this digital money is regulated by the nation's monetary authority or central bank. CBDC can offer several features such as efficiency, transparency, or financial inclusion [4, 9]. Researchers have discussed that (in CBDC) users transactions' privacy and regulatory oversight can coexist, e.g., in [15, 41]. In such a setting, users' amount of payments and even with whom they transact can remain confidential, while various regulations can be encoded into cryptographic algorithms run on users' transaction history by authorities, e.g., to ensure compliance with "Anti-Money Laundering". CBDC is still in its infancy and has not been adopted by banks yet. The adoption of such a model requires fundamental changes to and further digitalisation of the current banking infrastructure. Therefore, unlike CBDC which is more futuristic, the focus of our work is on the existing online banking systems. Nevertheless, when CBDC becomes mainstream, APP frauds might happen to the CBDC users too. In this case, (a variant of) our result can be integrated into such a framework to protect APP frauds victims.

9 CONCLUSION AND FUTURE RESEARCH

An APP fraud takes place when fraudsters deceive a victim to make a payment to a bank account controlled by the fraudsters. Although APP frauds have been growing at a concerning rate, the victims are not receiving a sufficient level of protection and the reimbursement rate is still low. Authorities and regulators have provided guidelines to prevent APP frauds occurrence and improve victims' protection, but these guidelines are still vague and open to interpretation. In this work, to facilitate APP frauds victims' reimbursement, we proposed the notion of "Payment with Dispute Resolution" (PwDR). We identified a set of vital properties that a PwDR scheme should possess and formally defined them. Moreover, we proposed a candidate construction and studied its cost via asymptotic and concrete runtime evaluation. Our cost analysis indicated that the construction is efficient.

Future research can investigate and identify key factors that improve the effectiveness of banks warnings. Furthermore, with the increase in the popularity of alternative payment platforms (e.g., CBDC or cryptocurrency), it is likely that fraudsters will target their users. Hence, another interesting future research direction would be to design secure dispute resolution protocols to protect APP frauds victims in these platforms as well.

REFERENCES

[1] A P20 Fraud and Criminal Transactions Working Group Paper. 2021. Best Practice Approaches For Combating Payee Scams. (2021). https://static1.squarespace.com/static/5efcc6dae323db37b4d01d19/t/60958e7453a1e0728b445470/1620414069082/P20+Report+-+Best+Practice+Approaches+For+Combating+Payee+Scams.pdf.

[2] Aydin Abadi, Steven J. Murdoch, and Thomas Zacharias. 2021. Recurring Contingent Payment for Proofs of Retrievability. Cryptology ePrint Archive, Report 2021/1145. https://ia.cr/2021/1145.

[4] Sarah Allen, Srđjan Čapkun, Ittay Eyal, Giulia Fanti, Bryan A Ford, James Grimmelmann, Ari Juels, Kari Kostiainen, Sarah Meiklejohn, Andrew Miller, Karl Wust, and Fan Zhang. 2020. *Design choices for central bank digital currency: Policy and technical considerations.* Technical Report. National Bureau of Economic Research.

[5] Ross Anderson et al. 2007. Closing the phishing hole–fraud, risk and nonbanks. In *Federal Reserve Bank of Kansas City–Payment System Research Conferences*. 41–56.

[6] 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).

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

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

[9] Bank of Canada, European Central Bank, Bank of Japan, Sveriges Riksbank, Swiss National Bank, Bank of England, Board of Governors Federal Reserve System, Bank for International Settlements. 2020. *Central bank digital currencies: foundational principles and core features.* Technical Report. https://www.bis.org/publ/othp33.pdf.

[10] 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).

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

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

[13] Prosenjit Bose, Hua Guo, Evangelos Kranakis, Anil Maheshwari, Pat Morin, Jason Morrison, Michiel H. M. Smid, and Yihui Tang. 2008. On the false-positive rate of Bloom filters. *Inf. Process. Lett.* (2008).

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

[15] David Chaum, Christian Grothoff, and Thomas Moser. 2021. How to Issue a Central Bank Digital Currency. *CoRR* abs/2103.00254 (2021). arXiv:2103.00254 https://arxiv.org/abs/2103.00254

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

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

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

[19] Adam French. 2016. Which? makes scams super-complaint-Banks must protect those tricked into a bank transfer. (2016). https://www.which.co.uk/news/2016/09/which-makes-scams-super-complaint-453196/.

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

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

[22] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, Second Edition.* CRC Press. https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269

[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] 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.

[26] David McIlroy and Ruhi Sethi-Smith. 2021. Prospects for bankers' liability for authorised push payment fraud. *Butterworths Journal of International Banking and Financial Law* (2021). https://www.forumchambers.com/wp-content/uploads/2021/03/Article-2-Smith.1-1.pdf.

[27] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system.* Technical Report.

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

[29] 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).

[30] Torben P. Pedersen. 1991. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO*.

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

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

[33] 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).
[34] 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.
[35] The Federal Reserve. 2020. Fraud Classifier. (2020). https://fedpaymentsimprovement.org/wp-content/uploads/fraudclassifier-industry-adoption-roadmap.pdf.
[36] 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.
[37] 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.
[38] UK Finance. 2021. 2021 Half Year Fraud Update. https://www.ukfinance.org.uk/system/files/Half-year-fraud-update-2021-FINAL.pdf.
[39] UK Finance. 2021. THE DEFINITIVE OVERVIEW OF PAYMENT INDUSTRY FRAUD. https://www.ukfinance.org.uk/system/files/Fraud%20The%20Facts%202021-%20FINAL.pdf.
[40] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* (2014).
[41] Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. 2019. PRCash: Fast, Private and Regulated Transactions for Digital Currencies. In *FC*.

A 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: $r \xleftarrow{\$} \{0,1\}^\lambda$. In the end of the commit phase, the commitment $\mathtt{Com}_x$ is sent to the receiver. In the open phase, the sender sends the opening $\ddot{x} := (x, r)$ to the receiver who verifies its correctness: $\mathtt{Ver}(\mathtt{Com}_x, \ddot{x}) \stackrel{?}{=} 1$ and accepts if the output is 1. A commitment scheme must satisfy two properties: (a) *hiding*: it is infeasible for an adversary (i.e., the receiver) to learn any information about the committed message x, until the commitment $\mathtt{Com}_x$ is opened, and (b) *binding*: it is infeasible for an adversary (i.e., the sender) to open a commitment $\mathtt{Com}_x$ to different values $\ddot{x}' := (x', r')$ than that was used in the commit phase, i.e., infeasible to find $\ddot{x}'$, *s.t.* $\mathtt{Ver}(\mathtt{Com}_x, \ddot{x}) = \mathtt{Ver}(\mathtt{Com}_x, \ddot{x}') = 1$, where $\ddot{x} \neq \ddot{x}'$. There exist efficient non-interactive commitment schemes both in (a) the standard model, e.g., Pedersen scheme [30], and (b) the random oracle model using the well-known hash-based scheme such that committing is : $\mathtt{H}(x||r) = \mathtt{Com}_x$ and $\mathtt{Ver}(\mathtt{Com}_x, \ddot{x})$ requires checking: $\mathtt{H}(x||r) \stackrel{?}{=} \mathtt{Com}_x$, where $\mathtt{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is a collision resistant hash function; i.e., the probability to find x and x' such that $\mathtt{H}(x) = \mathtt{H}(x')$ is negligible in the security parameter λ.

B BLOOM FILTER

A Bloom filter [11] is a compact data structure for probabilistic efficient elements' membership checking. A Bloom filter is an array of m bits that are initially all set to zero. It represents n elements. A Bloom filter comes along with k independent hash functions. To insert an element, all the hash values of the element are computed and their corresponding bits in the filter are set to 1. To check an element's membership, all its hash values are re-computed and checked whether all are set to one in the filter. If all the corresponding bits are one, then the element is probably in the filter; otherwise, it is not. In Bloom filters false positives are possible, i.e. it is possible that an element is not in the set, but the membership query shows that it is. According to [13], the upper bound of the false positive probability is: $q = p^k(1 + O(\frac{k}{p}\sqrt{\frac{\ln m - k \ln p}{m}}))$, where p is the probability that a particular bit in the filter is set to 1 and calculated as: $p = 1 - (1 - \frac{1}{m})^{kn}$. The efficiency of a Bloom filter depends on m and k. The lower bound of m is $n \log_2 e \cdot \log_2 \frac{1}{q}$, where e is the base of natural logarithms, while the optimal number of hash functions is $\log_2 \frac{1}{q}$, when m is optimal. In this paper, we only use optimal k and m. In practice, we would like to have a predefined acceptable upper bound on false positive probability, e.g. $q = 2^{-40}$. Thus, given q and n, we can determine the rest of the parameters.

C FULL VERSION OF DEFINITION OF PAYMENT WITH DISPUTE RESOLUTION SCHEME

In this section, we present a full formal definition of a PwDR scheme. Note, this section has overlapping content with Section 5; however, to keep this section self-contained, we let it happen. Below, we first provide the scheme's syntax. Then, we formally define its correctness and security properties.

Definition C.1. A payment with dispute resolution scheme includes the following algorithms PwDR := (keyGen, bankInit, customerInit, genUpdateRequest, insertNewPayee, genWarning, genPaymentRequest, makePayment, genComplaint, verComplaint, resDispute). It involves six types of entities; namely, bank $\mathcal{B}$, customer $\mathcal{C}$, smart contract $\mathcal{S}$, certificate generator $\mathcal{G}$, set of arbiters $\mathcal{D} : \{\mathcal{D}_1, ..., \mathcal{D}_n\}$, and dispute resolver $\mathcal{DR}$. Below, we define these algorithms.

- $\mathsf{keyGen}(1^\lambda) \rightarrow (sk, pk)$. It is a probabilistic algorithm run independently by $\mathcal{G}$ and one of the arbiters, $\mathcal{D}_j$. It takes as input a security parameter 1^λ. It outputs a pair of secret keys $sk := (sk_{\mathcal{G}}, sk_{\mathcal{D}})$ and public keys $pk := (pk_{\mathcal{G}}, pk_{\mathcal{D}})$. The public key pair, pk, is sent to all participants.

- $\mathsf{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$. It is run by $\mathcal{B}$. It takes as input security parameter 1^λ. It allocates private parameters to $\ddot{\pi}_1$ and $\ddot{\pi}_2$. It generates 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. Given a value and its witness anyone can check if they match. It also generates a set of (additional) public parameters, pp, one of which is e that is a threshold parameter. It also generates an empty list, $\boldsymbol{l}$. It outputs T, pp and $\boldsymbol{l}$. $\mathcal{B}$ sends $(\ddot{\pi}_1, \ddot{\pi}_2)$ to $\mathcal{C}$ and sends $(g_1, g_2, pp, \boldsymbol{l})$ to $\mathcal{S}$.

- $\mathsf{customerInit}(1^\lambda, T, pp) \rightarrow a$. It is a deterministic algorithm run by $\mathcal{C}$. It takes as input security parameter 1^λ, token T, and set pp of public parameters. It checks the correctness of the elements in T and pp. If the checks pass, it outputs 1. Otherwise, it outputs 0.

- $\mathsf{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(\mathcal{C})}$. It is a deterministic algorithm run by $\mathcal{C}$. It takes as input token T, new payee's detail f and empty payees' list $\boldsymbol{l}$. It generates $m_1^{(\mathcal{C})}$ which is an update request to the payees' list. It uses $T_1 \in T$ and encoding algorithm $\mathsf{Encode}(T_1, .)$ to encode $m_1^{(\mathcal{C})}$ which results in $\hat{m}_1^{(\mathcal{C})}$. It outputs $\hat{m}_1^{(\mathcal{C})}$. $\mathcal{C}$ sends the output to $\mathcal{S}$.

- $\mathsf{insertNewPayee}(\hat{m}_1^{(\mathcal{C})}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$. It is a deterministic algorithm run by $\mathcal{S}$. It takes as input $\mathcal{C}$'s encoded update request $\hat{m}_1^{(\mathcal{C})}$, and $\mathcal{C}$'s payees' list $\boldsymbol{l}$. It inserts the new payee's detail into $\boldsymbol{l}$ and outputs an updated list, $\hat{\boldsymbol{l}}$.

- $\mathsf{genWarning}(T, \hat{\boldsymbol{l}}, \text{aux}) \rightarrow \hat{m}_1^{(\mathcal{B})}$. It is run by $\mathcal{B}$. It takes as input token T, $\mathcal{C}$'s encoded payees' list $\hat{\boldsymbol{l}}$ and auxiliary information: aux, e.g., a set of policies. Using $T_1 \in T$, it decodes and checks all elements of the list, e.g., whether they comply with the policies. If the check passes, it sets $m_1^{(\mathcal{B})}$ = "pass"; otherwise, it sets $m_1^{(\mathcal{B})} = \textit{warning}$, where the *warning* is a string containing a warning detail along with the string "warning". It uses T_1 and $\mathsf{Encode}(T_1, .)$ to encode $m_1^{(\mathcal{B})}$ which yields $\hat{m}_1^{(\mathcal{B})}$. It outputs $\hat{m}_1^{(\mathcal{B})}$. $\mathcal{B}$ sends $\hat{m}_1^{(\mathcal{B})}$ to $\mathcal{S}$.

- $\mathsf{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(\mathcal{C})}$. It is run by $\mathcal{C}$. It takes as input token T, a payment detail in_f, encoded payees' list $\hat{\boldsymbol{l}}$, and encoded warning message, $\hat{m}_1^{(\mathcal{B})}$. Using $T_1 \in T$, it decodes $\hat{\boldsymbol{l}}$ and $\hat{m}_1^{(\mathcal{B})}$ yielding $\boldsymbol{l}$ and $m_1^{(\mathcal{B})}$ respectively. It checks the warning. It sets $m_2^{(\mathcal{C})} = \phi$, if it does not want to proceed. Otherwise, it sets $m_2^{(\mathcal{C})}$ according to the content of in_f and $\boldsymbol{l}$ (e.g., the amount of payment and payee's detail). It uses T_1 and $\mathsf{Encode}(T_1, .)$ to encode $m_2^{(\mathcal{C})}$ resulting in $\hat{m}_2^{(\mathcal{C})}$. It outputs $\hat{m}_2^{(\mathcal{C})}$. $\mathcal{C}$ sends $\hat{m}_2^{(\mathcal{C})}$ to $\mathcal{S}$.

- $\mathsf{makePayment}(T, \hat{m}_2^{(\mathcal{C})}) \rightarrow \hat{m}_2^{(\mathcal{B})}$. It is a deterministic algorithm run by $\mathcal{B}$. It takes as input token T, and encoded payment detail $\hat{m}_2^{(\mathcal{C})}$. Using $T_1 \in T$, it decodes $\hat{m}_2^{(\mathcal{C})}$ and checks the result's validity, e.g., ensures it is well-formed or $\mathcal{C}$ has enough credit. If the check passes, it makes the payment and sets $m_2^{(\mathcal{B})}$ = "paid". Otherwise, it sets $m_2^{(\mathcal{B})} = \phi$. It uses T_1 and $\mathsf{Encode}(T_1, .)$ to encode $m_2^{(\mathcal{B})}$ yielding $\hat{m}_2^{(\mathcal{B})}$. It outputs $\hat{m}_2^{(\mathcal{B})}$. $\mathcal{B}$ sends $\hat{m}_2^{(\mathcal{B})}$ to $\mathcal{S}$.

- $\texttt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, \text{aux}_f) \rightarrow (\hat{z}, \hat{\ddot{\pi}})$. It is run by C. It takes as input the encoded warning message $\hat{m}_1^{(\mathcal{B})}$, encoded payment message $\hat{m}_2^{(\mathcal{B})}$, token T, public key pk, and auxiliary information aux_f. It initially sets fresh strings (z_1, z_2, z_3) to null. Using $T_1 \in T$, it decodes $\hat{m}_1^{(\mathcal{B})}$ and $\hat{m}_2^{(\mathcal{B})}$ and checks the results' content. If it wants to complain that (i) "pass" message should have been a warning or (ii) no message was provided, it sets z_1 to "challenge message". If its complaint is about the warning's effectiveness, it sets z_2 to a combination of an evidence $u \in aux_f$, 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. In certain cases, the certificate might be empty. If its complaint is about the payment, it sets z_3 to "challenge payment". It uses T_1 and $\texttt{Encode}(T_1, .)$ to encode $z := (z_1, z_2, z_3)$ and uses $pk_{\mathcal{D}}$ and another encoding algorithm $\bar{\texttt{Encode}}(pk_{\mathcal{D}}, .)$ to encode $\ddot{\pi} := (\ddot{\pi}_1, \ddot{\pi}_2) \in T$. This results in $\hat{z}$ and $\hat{\ddot{\pi}}$ respectively. It outputs $(\hat{z}, \hat{\ddot{\pi}})$. C sends the pair to $\mathcal{S}$.

- $\texttt{verComplaint}(\hat{z}, \hat{\ddot{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j$. It is run by every arbiter $\mathcal{D}_j$. It takes as input C's encoded complaint $\hat{z}$, encoded private parameters $\hat{\ddot{\pi}}$, the tokens' public parameters $g := (g_1, g_2)$, encoded messages $\hat{\boldsymbol{m}} = [\hat{m}_1^{(C)}, \hat{m}_2^{(C)}, \hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}]$, encoded payees' list $\hat{\boldsymbol{l}}$, the arbiter's index j, secret key $sk_{\mathcal{D}}$, auxiliary information aux, and set of public parameters pp. It uses $sk_{\mathcal{D}}$ to decode $\hat{\ddot{\pi}}$ that yields $\ddot{\pi} := (\ddot{\pi}_1, \ddot{\pi}_2)$. It uses $\ddot{\pi}_1$ to decode $\hat{z}$, $\hat{\boldsymbol{m}}$, and $\hat{\boldsymbol{l}}$ that results in $z := (z_1, z_2, z_3)$, $\boldsymbol{m}$, and $\boldsymbol{l}$ respectively. It checks if $\ddot{\pi}_i$ matches g_i. If the check fails, it aborts. It checks if $m_1^{(C)}$ and $m_2^{(C)}$ are non-empty; it aborts if the checks fail. It sets fresh parameters $(w_{1,j}, w_{2,j}, w_{3,j}, w_{4,j})$ to ϕ. If "challenge message" $\in z_1$, given $\boldsymbol{l}$, it checks whether "pass" message (in $m_1^{(\mathcal{B})}$) was given correctly or the missing message did not play any role in preventing the fraud. If either checks passes, it sets $w_{1,j} = 0$; otherwise, it sets $w_{1,j} = 1$. If "challenge warning" $\in z_2$, it verifies the certificate in z_2. If it is invalid, it sets $w_{3,j} = 0$. If it is valid, it sets $w_{3,j} = 1$. It determines the effectiveness of the warning (in $m_1^{(\mathcal{B})}$), by running an algorithm which determines that, i.e., $\texttt{checkWarning}(.) \in aux$. If it is effective, i.e., $\texttt{checkWarning}(m_1^{(\mathcal{B})}) = 1$, it sets its verdict to 0, i.e., $w_{2,j} = 0$; otherwise, it sets $w_{2,j} = 1$. If "challenge payment" $\in z_3$, it checks if the payment was made (with the help of $m_2^{(\mathcal{B})}$). If the check passes, it sets $w_{4,j} = 1$; otherwise, it sets $w_{4,j} = 0$. It uses $\ddot{\pi}_2$ to encode $\boldsymbol{w}_j = [w_{1,j}, w_{2,j}, w_{3,j}, w_{4,j}]$ yielding $\hat{\boldsymbol{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]$. It outputs $\hat{\boldsymbol{w}}_j$. $\mathcal{D}_j$ sends $\hat{\boldsymbol{w}}_j$ to $\mathcal{S}$.

- $\texttt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v}$. It is a deterministic algorithm run by $\mathcal{DR}$. It takes as input token T_2, arbiters' encoded verdicts $\hat{\boldsymbol{w}} = [\hat{\boldsymbol{w}}_1, ..., \hat{\boldsymbol{w}}_n]$, and public parameters set pp. It checks if the token's parameters match. If the check fails, it aborts. It uses $\ddot{\pi}_2 \in T_2$ to decode $\hat{\boldsymbol{w}}$ and from the result it extracts final verdicts $\boldsymbol{v} = [v_1, ..., v_4]$. The extraction procedure ensures each v_i is set to 1 only if at least e arbiters' original verdicts (i.e., $w_{i,j,}$) is 1, where $e \in pp$. It outputs $\boldsymbol{v}$. If $v_4 = 1$ and (i) either $v_1 = 1$ (ii) or $v_2 = 1$ and $v_3 = 1$, then C is reimbursed.

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

Definition C.2 (Correctness). A PwDR scheme is correct if the key generation algorithm produces keys $\texttt{keyGen}(1^\lambda) \rightarrow (sk, pk)$ such that for any payee's detail f, payment's detail in_f, and auxiliary information (aux, aux_f), if $\texttt{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$, $\texttt{customerInit}(1^\lambda, T, pp) \rightarrow a$, $\texttt{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$, $\texttt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$, $\texttt{genWarning}(T, \hat{\boldsymbol{l}}, \text{aux}) \rightarrow \hat{m}_1^{(\mathcal{B})}$, $\texttt{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$, $\texttt{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$, $\texttt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, \text{aux}_f) \rightarrow (\hat{z}, \hat{\ddot{\pi}})$, $\forall j \in [n] : \Big(\texttt{verComplaint}(\hat{z}, \hat{\ddot{\pi}}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j\Big)$, $\texttt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v}$, then $(z_1 = z_2 = z_3 = \phi) \wedge (\boldsymbol{v} = 0)$, where $g := (g_1, g_2) \in T$, $\hat{\boldsymbol{m}} = [\hat{m}_1^{(C)}, \hat{m}_2^{(C)}, \hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}]$, $\hat{\boldsymbol{w}} = [\hat{\boldsymbol{w}}_1, ..., \hat{\boldsymbol{w}}_n]$, and $z := (z_1, z_2, z_3)$ is the result of decoding $\hat{z}$.

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

was effective, or (d) make $\mathcal{DR}$ believe that the warning message was ineffective although only less than threshold $\mathcal{D}_j$s believe that, or (e) convince $\mathcal{D}_j$s to accept an invalid certificate, or (f) make $\mathcal{DR}$ believe that at least threshold $\mathcal{D}_j$s accepted the certificate although they did not, except for a negligible probability.

Definition C.3 (Security against a malicious victim). A PwDR scheme is secure against a malicious victim, if for any security parameter λ, auxiliary information aux, and probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$, such that for an experiment $\mathsf{Exp}_1^{\mathcal{A}}$:

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

$\mathtt{keyGen}(1^\lambda) \rightarrow (sk, pk)$
$\mathtt{bankInit}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l})$
$\mathcal{A}(1^\lambda, T, pp, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$
$\mathtt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$
$\mathtt{genWarning}(T, \hat{\boldsymbol{l}}, \mathrm{aux}) \rightarrow \hat{m}_1^{(\mathcal{B})}$
$\mathcal{A}(T, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$
$\mathtt{makePayment}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$
$\mathcal{A}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk) \rightarrow (\hat{z}, \hat{\pi})$
$\forall j, j \in [n]:$
$\left(\mathtt{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]\right)$
$\mathtt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v} = [v_1, ..., v_4]$

it holds that:

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

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

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

Definition C.4 (Security against a malicious bank). A PwDR scheme is secure against a malicious bank, if for any security parameter λ, auxiliary information aux, and probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function $\mu(\cdot)$, such that for an experiment $\mathsf{Exp}_2^{\mathcal{A}}$:

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

$\mathtt{keyGen}(1^\lambda) \rightarrow (sk, pk)$
$\mathcal{A}(1^\lambda) \rightarrow (T, pp, \boldsymbol{l}, f, in_f, \mathrm{aux}_f)$
$\mathtt{customerInit}(1^\lambda, T, pp) \rightarrow a$
$\mathtt{genUpdateRequest}(T, f, \boldsymbol{l}) \rightarrow \hat{m}_1^{(C)}$
$\mathtt{insertNewPayee}(\hat{m}_1^{(C)}, \boldsymbol{l}) \rightarrow \hat{\boldsymbol{l}}$
$\mathcal{A}(T, \hat{\boldsymbol{l}}, \mathrm{aux}) \rightarrow \hat{m}_1^{(\mathcal{B})}$
$\mathtt{genPaymentRequest}(T, in_f, \hat{\boldsymbol{l}}, \hat{m}_1^{(\mathcal{B})}) \rightarrow \hat{m}_2^{(C)}$
$\mathcal{A}(T, \hat{m}_2^{(C)}) \rightarrow \hat{m}_2^{(\mathcal{B})}$
$\mathtt{genComplaint}(\hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}, T, pk, \mathrm{aux}_f) \rightarrow (\hat{z}, \hat{\pi})$
$\forall j, j \in [n]:$
$\left(\mathtt{verComplaint}(\hat{z}, \hat{\pi}, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, j, sk_{\mathcal{D}}, aux, pp) \rightarrow \hat{\boldsymbol{w}}_j = [\hat{w}_{1,j}, \hat{w}_{2,j}, \hat{w}_{3,j}, \hat{w}_{4,j}]\right)$
$\mathtt{resDispute}(T_2, \hat{\boldsymbol{w}}, pp) \rightarrow \boldsymbol{v} = [v_1, ..., v_4]$

it holds that:

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

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

A careful reader may ask why the following two conditions (some forms of which were in the events of Definition C.3) are not added to the above events list: (a) $\mathcal{B}$ makes at least threshold committee members conclude that it has provided a warning, although $\mathcal{B}$ has not (i.e., $m_1^{(\mathcal{B})} \neq warning \wedge \sum_{j=1}^{n} w_{1,j} < e$), and (b) $\mathcal{B}$ persuades at least threshold $\mathcal{D}_j$s to conclude that the warning was effective although it was not (i.e., $\mathtt{checkWarning}(m_1^{(\mathcal{B})}) = 0 \wedge \sum_{j=1}^{n} w_{2,j} < e$). The answer is that $\mathcal{B}$ does not generate a complaint and interact directly with $\mathcal{D}_j$s; therefore, we do not need to add these two events to the above events' list. Now we move on to privacy. Informally, a PwDR is privacy-preserving if it protects the privacy of (1) the customers', bank's, and arbiters' messages (except public parameters) from non-participants of the protocol, including other customers, and (2) each arbiter's verdict from $\mathcal{DR}$ which sees the final verdict.

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

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

, ,

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

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

it holds that:

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

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

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

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

it holds that:

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

where $g := (g_1, g_2) \in T$, $\hat{\boldsymbol{m}} = [\hat{m}_1^{(C)}, \hat{m}_2^{(C)}, \hat{m}_1^{(\mathcal{B})}, \hat{m}_2^{(\mathcal{B})}]$, $\hat{\boldsymbol{w}} = [\hat{\boldsymbol{w}}_1, .., \hat{\boldsymbol{w}}_n]$, input $:= (1^\lambda, aux)$, $T_1 \in T$, $pk_{\mathcal{D}} \in pk$, $sk_{\mathcal{D}} \in sk$, n is the total number of arbiters. Moreover, Pr' is defined as follows. Let arbiter $\mathcal{D}_i$ output 0 and 1 with probabilities $Pr_{i,0}$ and $Pr_{i,1}$ respectively. Then, Pr' is defined as $Max\{Pr_{1,0}, Pr_{1,1}, ..., Pr_{n,0}, Pr_{n,1}\}$. In the above privacy definition, the probability is taken over the uniform choice of sk, the probability that each $\mathcal{D}_j$ outputs 0 or 1, the randomness used in the blockchain's primitives, the randomness used during the encoding, and the randomness of $\mathcal{A}_1$ in $\mathsf{Exp}_3^{\mathcal{A}_1}$ and $\mathcal{A}_2$ in $\mathsf{Exp}_4^{\mathcal{A}_2}$.

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

D FURTHER DISCUSSION ON VARIANT 1 ENCODING-DECODING PROTOCOL

In this section, we first formally state our observation on which Variant 1 encoding-decoding protocol relies and then prove it. After that, we explain why this variant meets the three properties we laid out in Section 6.4, i.e., it should (1) generate unlinkable verdicts, (2) not require arbiters to interact with each other for each customer, and (3) be efficient.

Theorem D.1. *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 $\tilde{s} = \underbrace{s_1 \oplus s_2 \oplus ... \oplus s_j}_{|S'|}$, where $s_i \in S'$. Since each s_i is a uniformly random value, the XOR of them is a uniformly random value too. That means values $\tilde{s}$ and s have identical distribution. Thus, $\tilde{s}$ reveals nothing in this case as well. Also, it is not hard to see that the combination of the above two cases reveals nothing too, i.e., $\bar{s} \oplus \tilde{s}$ and s have identical distribution. □

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 arbiter that voted 1 and the case where multiple arbiters voted 1, as in both cases $\mathcal{I}$ extracts only a single random value, which reveals nothing about the number of arbiters which voted 0 or 1 (due to Theorem D.1). Note, the protocols' correctness holds with an overwhelming probability, i.e., $1 - \frac{1}{2^\lambda}$. Specifically, if two arbiters 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., at most $\frac{1}{2^\lambda}$. It is evident that this variant meets property (2) as the arbiters 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.

E GENERIC VERDICT ENCODING-DECODING PROTOCOL

In this section, we present the generic verdict encoding-decoding protocols (i.e., GPVE and GFVD) in detail, in Figures 5 and 6. They let a semi-honest third party $\mathcal{I}$ find out if at least e arbiters voted 1, where e can be any integer in the range $[1, n]$.

As Figure 5 indicates (and as we discussed in Section 6.4) 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 $\mathcal{I}$. Nevertheless, instead of inserting the combinations into a Bloom filter, we could *hash* the combinations and give the hash values to $\mathcal{I}$. But, using a Bloom filter allows us to save considerable communication costs. Now, we provide 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.

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

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

Arbiter $\mathcal{D}_j$ 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 the range $[e, n]$, computes the combinations (without repetition) of i elements from set $\{\alpha_1, ..., \alpha_n\}$. In the case where multiple elements are taken at a time (i.e., $i > 1$), the elements are XORed with each other. Let $W = \{(\alpha_1 \oplus ... \oplus \alpha_e), (\alpha_2 \oplus ... \oplus \alpha_{e+1}), ..., (\alpha_1 \oplus ... \oplus \alpha_n)\}$ 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 $(\bar{w}_j, \mathsf{BF})$.

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

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

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

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

(1) combines all arbiters' 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.

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

F VARIANT 2 ENCODING-DECODING PROTOCOL'S MAIN THEOREM AND PROOF

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

Theorem F.1. *Let set $S = \{s_1, ..., s_m\}$ be a set of random values picked uniformly from $\mathbb{F}_p$, 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, 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 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. Thus, the only information BF reveals is the public one. □

G FURTHER DISCUSSION ON THE VERDICT ENCODING-DECODING PROTOCOL

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

Is there any e-voting protocol, 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 has been 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 arbiters (or voters) which are trusted and can interact with each other once; whereas, the latter (e-voting) protocols are in a more generic setting where there is a large number of voters, some of which might be malicious, and they are not required to interact with each other.

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

H SECURITY ANALYSIS OF THE PWDR PROTOCOL

In this section, we prove the security of the PwDR protocol. We first present the PwDR protocol's main theorem.

Theorem H.1. *The PwDR protocol (presented in Section 6.5) is secure, with regard to definition C.6, if the digital signature is existentially unforgeable under chosen message attacks, the blockchain, SAP, and pseudorandom function* PRF(.) *are secure, the encryption schemes are semantically secure, and the correctness of verdict encoding-decoding protocols (i.e., PVE and FVD) holds.*

To prove the above theorem, we show that the PwDR scheme satisfies all security properties defined in Section 5. We first prove that it meets security against a malicious victim.

Lemma H.2. *If the digital signature is existentially unforgeable under chosen message attacks, and the SAP and blockchain are secure, then the PwDR scheme is secure against a malicious victim, with regard to Definition C.3.*

, ,

Proof. First, we focus on event I : $\Big((m_1^{(\mathcal{B})} = warning) \wedge (\sum_{j=1}^{n} w_{1,j} \geq e)\Big)$ which considers the case where $\mathcal{B}$ has provided a warning message but $\mathcal{C}$ manages to convince at least threshold arbiters to set their verdicts to 1, that ultimately results in $\sum_{j=1}^{n} w_{1,j} \geq e$. We argue that the adversary's success probability in this event is negligible in the security parameter. In particular, due to the security of SAP, $\mathcal{C}$ cannot convince an arbiter 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 arbiter 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 arbiter with a high probability. Next, we consider event II : $\Big((\sum_{j=1}^{n} w_{1,j} < e) \wedge (v_1 = 1)\Big)$ that captures the case where only less than threshold arbiters 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 threshold arbiters approved that. We argue that the probability that this event occurs is negligible in the security parameter too. Specifically, due to the security of the SAP, $\mathcal{C}$ cannot persuade (a) an arbiter 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. Specifically, 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 : $\Big((\texttt{checkWarning}(m_1^{(\mathcal{B})}) = 1) \wedge (\sum_{j=1}^{n} w_{2,j} \geq e)\Big)$. It captures the case where $\mathcal{B}$ has provided an effective warning message but $\mathcal{C}$ manages to make at threshold arbiters 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 arbiter to accept a different decryption key other than what was agreed between $\mathcal{C}$ and $\mathcal{B}$ in the initiation phase. Therefore, all arbiters will receive the original message of $\mathcal{B}$, including the effective warning message, except a negligible probability, $\mu(\lambda)$. Now, we consider event IV : $\Big((\sum_{j=1}^{n} w_{2,j} < e) \wedge (v_2 = 1)\Big)$ which captures the case where at least threshold arbiters approved that the warning message was effective but the final verdict that $\mathcal{DR}$ extracts implies that they approved the opposite. The security argument of event II applies to this event as well. In short, due to the security of the SAP, $\mathcal{C}$ cannot persuade an arbiter to accept a different encryption key, and cannot convince $\mathcal{DR}$ to accept a different decryption key other than what was initially agreed between $\mathcal{C}$ and $\mathcal{B}$, except a negligible probability, $\mu(\lambda)$. Now, we analyse event V : $\Big(u \notin Q \wedge \texttt{Sig.ver}(pk, u, sig) = 1\Big)$. This even captures the case where the malicious victim comes up with a valid signature/certificate on a message that has never been queried to the signing oracle. Nevertheless, due to the existential unforgeability of the digital signature scheme, the probability that such an event occurs is negligible, $\mu(\lambda)$. Next, we focus on event VI : $\Big((\sum_{j=1}^{n} w_{3,j} < e) \wedge (v_3 = 1)\Big)$ that considers the case where less than threshold arbiters indicated that the signature (in $\mathcal{C}$'s complaint) is valid, but the final verdict that $\mathcal{DR}$ extracts implies that at least threshold arbiters approved the signature. This means the adversary has managed to switch the verdicts of those arbiters which voted 0 to 1. However, the probability that this even occurs is negligible as well. Because, due to the SAP's security, $\mathcal{C}$ cannot convince an arbiter and $\mathcal{DR}$ to accept different encryption and decryption keys other than what was initially agreed between $\mathcal{C}$ and $\mathcal{B}$, except a negligible probability, $\mu(\lambda)$. Therefore, with a negligible probability the adversary can switch a verdict for 0 to the verdict for 1.

Moreover, a malicious $\mathcal{C}$ cannot frame an honest $\mathcal{B}$ for providing an invalid message by manipulating the smart contract's content, e.g., by replacing an effective warning with an ineffective one in $\mathcal{S}$, or excluding a warning from $\mathcal{S}$. In particular, 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 [20]. □

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

Lemma H.3. *If the SAP and blockchain are secure, and the correctness of verdict encoding-decoding protocols (i.e., PVE and FVD) holds, then the PwDR protocol is secure against a malicious bank, with regard to Definition C.4.*

Proof. We first focus on event I : $\Big((\sum_{j=1}^{n} w_{1,j} \geq e) \wedge (v_1 = 0)\Big)$ 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 threshold arbiters do not believe so. We argue that the probability that this event takes place is negligible in the security parameter. Because $\mathcal{B}$ cannot persuade $\mathcal{DR}$ to accept a different decryption key, e.g., $k' \in \ddot{\pi}'$, other than what was agreed between $\mathcal{C}$ 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 6.4, due to the correctness of the verdict encoding-decoding protocols, i.e., PVE and FVD, the probability that multiple representations of verdict 1 cancel out each other is negligible too, $\frac{1}{2^{\lambda}}$. Thus, event I occurs only with a negligible probability, $\mu(\lambda)$. To assert that events II : $\Big((\sum_{j=1}^{n} w_{2,j} \geq e) \wedge (v_2 = 0)\Big)$, III : $\Big((\sum_{j=1}^{n} w_{3,j} \geq e) \wedge (v_3 = 0)\Big)$, and IV : $\Big((\sum_{j=1}^{n} w_{4,j} \geq e) \wedge (v_4 = 0)\Big)$ occur only with a negligible probability, we can directly use the above argument provided for event I. To avoid repetition, we do not restate them in this proof. Moreover, a malicious $\mathcal{B}$ cannot frame an honest $\mathcal{C}$ for providing an invalid message by manipulating the smart contract's content, e.g., by replacing its valid signature with an invalid one or sending a message on its behalf, due to the security of the blockchain. □

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

Lemma H.4. *If the encryption schemes are semantically secure, and the SAP and encoding-decoding schemes (i.e., PVE and FVD) are secure, then the PwDR protocol is privacy-preserving with regard to Definition C.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 $\mathrm{Exp}_3^{\mathcal{A}_1}$, 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 $\mathcal{C}$'s and $\mathcal{B}$'s exchanged messages (e.g., payment requests or warning messages) and $\mathcal{D}_j$s verdicts. Furthermore, due to the semantical security of the symmetric key and asymmetric key encryption schemes, given ciphertext $(\hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}, \hat{\ddot{\pi}})$ 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, the adversary cannot link a verdict to a specific arbiter with a probability significantly better than the maximum probability, Pr', that an arbiter 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 arbiters' verdicts are 0. We conclude that, excluding the case where the all verdicts are 0, given $(T_2, pk, pp, g, \hat{\boldsymbol{m}}, \hat{\boldsymbol{l}}, \hat{z}, \hat{\ddot{\pi}}, \hat{\boldsymbol{w}}, \boldsymbol{v})$, adversary $\mathcal{A}_3$'s success probability in experiment $\mathrm{Exp}_4^{\mathcal{A}_2}$ to link a verdict to an arbiter is at most $Pr' + \mu(\lambda)$. □

Theorem H.5. *The PwDR protocol is secure according to Definition C.6.*

Proof. Due to Lemma H.2, the PwDR protocol is secure against a malicious victim. Also, due to lemmas H.3 and H.4 it is secure against a malicious bank and is privacy-preserving, respectively. Thus, it satisfies all the properties of Definition C.6, meaning that the PwDR protocol is indeed secure according to this definition. □

I EVALUATION

I.1 Asymptotic Cost Analysis

In this section, we present a detailed evaluation of the PwDR protocol's computation and communication complexity.

I.1.1 Computation Cost. We first analyse $\mathcal{C}$'s cost. In Phase 3, $\mathcal{C}$ invokes a hash function twice to check the correctness of the private statements' parameters. In Phase 4, it invokes the symmetric encryption once to encrypt its update request. In Phase 7, it invokes the symmetric encryption twice to decrypt $\mathcal{B}$'s warning message and to encrypt its payment request. In Phase 9, it runs the symmetric encryption three times to decrypt $\mathcal{B}$'s warning and payment messages and to encrypt its complaint. In the same phase, it invokes the asymmetric encryption once to encrypt the private statements' opening. Therefore, $\mathcal{C}$'s complexity is $O(1)$. Next, we analyse $\mathcal{B}$'s cost. In Phase 2, it invokes the hash function twice to commit to two statements. In Phase 6, it calls the symmetric key encryption once to encrypt its outgoing warning message. In Phase 8, it also invokes the symmetric key encryption once to encrypt the outgoing payment message. Thus, $\mathcal{B}$'s complexity is $O(1)$ too. Next, we analyse each arbiter's cost. In Phase 10, each $\mathcal{D}_j$ invokes the asymmetric key encryption once to decrypt the private statements' openings. It also invokes the hash function twice to verify the openings. It invokes the symmetric key encryption six times to decrypt $\mathcal{C}$'s and $\mathcal{B}$'s messages that were posted on $\mathcal{S}$ (this includes $\mathcal{C}$'s complaint). Recall, in the same phase, each arbiter encodes its verdict using a verdict encoding protocol. Now, we evaluate the verdict encoding complexity of each arbiter 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 arbiter $\mathcal{D}_j$, except $\mathcal{D}_n$, invokes the pseudorandom function once to encode its verdict. However, arbiter $\mathcal{D}_n$ invokes the pseudorandom function $n-1$ times and XORs the function's outputs with each other. Thus, in case (a), arbiter $\mathcal{D}_n$'s complexity is $O(n)$ while the ret of arbiters' complexity is $O(1)$. In case (b), every arbiter $\mathcal{D}_j$, except $\mathcal{D}_n$, invokes the pseudorandom function twice to encode its verdict. But, arbiter $\mathcal{D}_n$ invokes the pseudorandom function $n-1$ times and XORs the function's outputs with each other. It also invokes the pseudorandom function n times to generate all arbiters' representations of verdict 1. It computes all $y = \sum_{i=e}^{n} \frac{n!}{i!(n-i)!}$ combinations of the representations that meet the threshold which involves $O(y)$ XORs. It also inserts y elements into a Bloom filter that requires $O(y)$ hash function evaluations. So, in case (b), arbiter $\mathcal{D}_n$'s complexity is $O(y)$ while the rest of the arbiters' complexity is $O(1)$. To conclude, in Phase 10, arbiter $\mathcal{D}_n$'s complexity is either $O(n)$ or $O(y)$, while the rest of the arbiters' complexity is $O(1)$. Now, we analyse $\mathcal{DR}$'s cost in Phase 11. It invokes the hash function once to check the private statement's correctness. It also performs $O(n)$ symmetric key decryption to decrypt arbiters' encoded verdicts. Now, we evaluate the verdict decoding complexity of $\mathcal{DR}$ for two cases: (a) $e = 1$ and (b) $e \in (1, n]$. In the former case (in which FVD is invoked), it performs $O(n)$ XOR to combine all verdicts. Its complexity is also $O(n)$ in the latter case (in which GFVD is invoked), with a small difference that it also invokes the Bloom filter's hash functions, to make a membership query to the Bloom filter. Thus, $\mathcal{DR}$'s complexity is $O(n)$.

I.1.2 Communication Cost. Now, we analyse the communication cost of the PwDR protocol. Briefly, $\mathcal{C}$'s complexity is $O(1)$ as in total it sends only six messages to other parties. Similarly, $\mathcal{B}$'s complexity is $O(1)$ as its total number of outgoing messages is only nine. Each arbiter $\mathcal{D}_j$ sends only four messages to the smart contract, so its complexity is $O(1)$. However, if GFVD is invoked, then arbiter $\mathcal{D}_n$ 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.

I.2 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 arbiters, i.e., arbiter $\mathcal{D}_n$, and the dispute resolver have non-constant complexities. These non-constant overheads were mainly imposed by the verdict inducing-decoding protocols. Therefore, to study these parties' runtime in the PwDR, we implemented both variants of the verdict encoding-decoding protocols (that were presented in Section 6.4). They were implemented in C++.[4] 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[5] for cryptographic

[4] See [7, 8] for the source code.
[5] https://www.cryptopp.com

primitives, the "GMP" library[6] for arbitrary precision arithmetics, and the "Bloom Filter" library [7]. In the experiment, we set the false-positive rate in a Bloom filter to 2^{-40} and the finite field size to 128 bits. Table 3 provides the runtime of the three types of parties for various numbers of arbiters 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 the table depicts, the runtime of $\mathcal{D}_n$ increases gradually from 0.019 to 10.15 milliseconds when the number of arbiters 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 arbiters increases. Nevertheless, the overall cost is very low. In particular, the highest runtime is only about 10 milliseconds which belongs to $\mathcal{D}_n$ when $n = 12$ and $e = 7$. It is also evident that the parties' runtime in the PVE and FVD protocols is much lower than their runtime in the GPVE and GFVD ones. To compare the parties' runtime, we also fixed the threshold to 6 (in GPVE and GFVD protocols) and ran the experiment for different values of n. Figure 7 summarises the result. As this figure indicates, the runtime of $\mathcal{D}_n$ and $\mathcal{DR}$ almost linearly grows when the number of arbiters increases. Moreover, $\mathcal{D}_n$ has a higher runtime than $\mathcal{DR}$ has, and its runtime growth is faster than that of $\mathcal{DR}$.

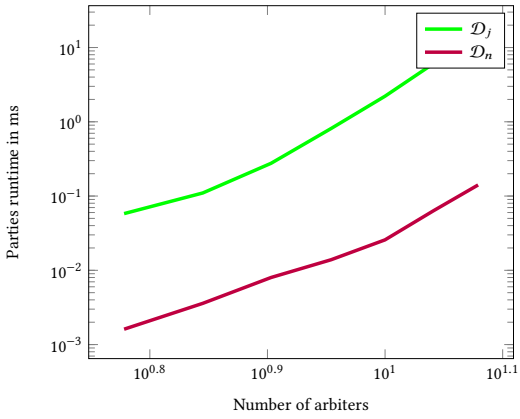

Fig. 7. Parties' runtime in the PwDR.

[6] https://gmplib.org
[7] http://www.partow.net/programming/bloomfilter/index.html