International Journal of Information Security (2024) 23:447–465
https://doi.org/10.1007/s10207-023-00744-5

REGULAR CONTRIBUTION

Pepal: Penalizing multimedia breaches and partial leakages

Easwar Vivek Mangipudi[1] · Krutarth Rao[2] · Jeremy Clark[3] · Aniket Kate[1]

Published online: 14 September 2023

Abstract
Storage of media files by users at a third party, like cloud services or escrows, is increasing every day along with the risk of stored files being leaked through breaches from third parties. In this article, we study the problem of handling either intentional or unintentional multimedia storage breaches by the entity hosting the data. To address the problem, we design the Pepal: protocol where the sender forwarding multimedia data to a receiver can penalize the receiver through loss of cryptocurrency even for partial data leakage. Pepal: achieves this by augmenting a blockchain on-chain smart contract between the two parties with an off-chain cryptographic protocol. The protocol involves a new primitive doubly oblivious transfer (DOT), which, when combined with robust watermarking and a claim-or-refund blockchain contract, provides the necessary framework for a provably secure protocol. Any public data leakage by the receiver leads to the sender learning the receiver's crypto-currency secret key, which allows him to transfer the claim-or-refund deposit of the receiver. The Pepal: protocol also ensures that the malicious sender cannot steal the deposit, even by leaking the original multimedia document in any form. We analyze our DOT-based design against partial adversarial leakages and show it to be robust against even small leakages. The prototype implementation of our Pepal: protocol shows our system to be efficient and easy to deploy in practice.

Keywords Data breach · Penalization · Oblivious transfer · Smart contract

1 Introduction

Data breach attacks on cloud platforms are increasing every year [2, 26, 29, 41], the reasons for which vary from compromises of ill-maintained data servers to careless data custodians. Although it has been observed and reported that 90% of the data breaches can be avoided with good security practices on the custodian's infrastructure [1], there is no evident decrease in the number. In these cases, taking legal action is not only expensive and time-consuming but it is also difficult to establish responsibility in today's geopolitically distributed data flows. It would be imperative to devise tools that would encourage the data hosts to adopt better security practices.

In this work, we introduce a complementary security mechanism that is inexpensive, automated, and not restricted by the geo-political boundaries to disincentivize data leakage. In particular, our goal is to make the data custodians more accountable through automatically enforceable monetary penalties resulting in immediate loss of funds. We ensure the penalization of even partial leakages (Pepal:) by enforcing a cryptocurrency claim-or-refund smart contract with the deposit made by the data receiver. Applicability scenarios for Pepal: contracts range from industrial media custodianship, data and software escrows, leaking privately shared personal data (pictures and other media files) of others on social media, and even to non-disclosure agreements between mutually distrusting entities [3].

Example Scenarios Data Escrow refers to the responsibility of safe storage of the data at a custodian [7]; any data breach typically results in criminal litigation against the custodian. However, legal action may be undesirable due to the uncertainty of recovering the payment (which increases if the winning party is owed court costs in addition to the actual

B Easwar Vivek Mangipudi
easwar.vivek@gmail.com

Krutarth Rao
raokrutarth@gmail.com

Jeremy Clark
j.clark@concordia.ca

Aniket Kate
aniket@purdue.edu

1 Purdue University, West Lafayette, USA

2 Hewlett Parckard, Spring, USA

3 Concordia University, Montreal, Canada

remedy) [21]. Pepal: can be useful in such circumstances by automating the monetary recovery procedure. We assume that the data sender and the custodian agree on an amount of money awarded to the owner should specified documents be demonstrably leaked. Towards automatically ensuring that the owner will receive the funds, this amount could take the form of a surety bond held in trust by a Bitcoin or other permissionless/permissioned blockchain-based cryptocurrency smart contract. Another interesting use case is users downloading paid media that should not be publicly shared online. The downloaders make a timed deposit, along with the actual payment, for an agreed time and value for the download. If no dishonest sharing happens, it will be returned to the downloader/customer else it will be forfeited.

Pepal: does not preclude the use of the court system, it simply complements it or shifts the responsibility of bringing legal action to the entity seeking to recover their bond. Allowing an escalation to court is essential as some disclosures are in the public interest (whistle-blowing) [14]. In some instances, a third party might pay the value of the bond for the information (news media, crowdfunding, etc.). We expect that the proposed mechanism encourages the parties involved to follow better security practices, and the proposed Pepal: protocol is a step in that direction.

Contributions We formalize and provide a solution direction to the problem of automatically settling intentional or unintentional data breaches with a blockchain smart contract, eschewing the traditional recourse of costly legal action. Our Pepal: protocol, which achieves it, is a crypto-augmented smart contract system obtaining an arbitrator-free settlement. It comprises a claim-or-refund smart contract, a robust watermarking scheme, a proposed cryptographic primitive – Doubly Oblivious Transfer (DOT), and a non-interactive zero-knowledge (NIZK) proof for mutually distrusting parties.

In our core protocol, the sender and receiver create a claim-or-refund transaction on Bitcoin [12, 15, 43] where an amount is deposited that can be spent at any time with a jointly signed transaction or spent after a period of time by an sender-only signed transaction. The document provided to the receiver has the receiver's signing (private) key embedded in it with a robust binary watermarking scheme that cannot be removed (or retrieved) by anyone except the embedding party. The challenging aspects of the Pepal: protocol involve arranging for the signing key to be embedded such that (1) the sender does not learn the value of the key at the time of embedding, (2) the receiver does not learn the document contents until the key is embedded, and (3) the sender is convinced the embedded key is the receiver's correct signing key. Within these constraints, to perform the embedding, the parties must jointly perform a two-party computation with their respective private inputs. Our novel DOT and committed receiver oblivious transfer (CROT) protocols, securely realize this two-party computation to ensure that the sender can retrieve the receiver's embedded key from the document if it leaks (widely enough to reach the sender) and spend the deposited cryptocurrency.

We implement the Pepal: system using the Relic library for the cryptographic primitives, a robust image watermarking scheme and claim-or-refund contract for Bitcoin. Given the prevalence of robust watermarking in the multimedia industry [4, 6], we find our Pepal: system easy to deploy. Our single-threaded implementation takes on average 1.73 seconds when a 1.3MB image is used as data for the transfer when the 256-bit key is embedded once.

Given the inherent non-cryptographic robustness guarantees of the robust watermarking system, we also analyze partial data disclosures. In particular, even when the receiver decides to reveal the document partially, our proposed DOT protocol ensures that the embedding party or the sender can retrieve significantly more bits than when the standard oblivious transfer is used. For example, when the receiver's 256-bit signing key is embedded 16 times in the multimedia document, even a 15% leakage of document blocks reveals roughly 235 bits of receiver's key to the sender with DOT; as opposed to roughly only 50 bits that are revealed when an oblivious transfer protocol is employed.

2 Solution overview

We consider a scenario where a sender wishes to forward a private multimedia document M to a receiver. The receiver is expected to hold a public key-secret key pair (pk, sk), where the key sk is a signing key of a (say) Bitcoin wallet corresponding to pk. Instead of the sender directly sending M to the receiver, we expect the sender and receiver to jointly compute a function $f((M, pk), sk)$, which should provide the receiver a version M_{sk} of M that has been tagged (or robustly watermarked) with the key sk. The protocol should abort (or not produce a meaningful M_{sk}) if sk from the receiver and pk from the sender are not matching key pair. At the end of the protocol, the sender does not learn sk or M_{sk} and the receiver does not learn any further information about M. A cryptocurrency wallet holds the receiver's escrow deposit for accountability.

We consider the problem in a mutually distrustful setting, and *either* the sender or the receiver can be malicious. A malicious sender can try to learn the signing key of the receiver to steal the deposit. When appropriate, he can also make the document public and try to accuse the receiver of dishonest disclosure. On the other hand, the malicious receiver can try to remove/replace the watermark from the obtained document and release the modified version to the public without revealing her key. In such an adversarial setting, we wish to satisfy the following privacy and integrity goals:

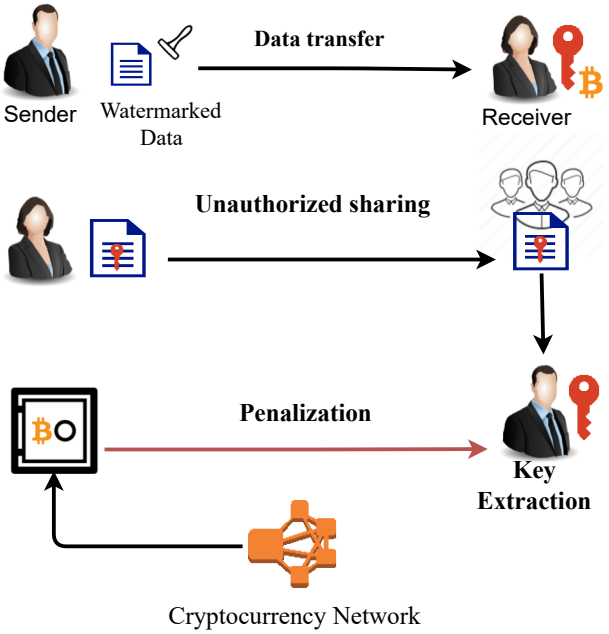


Fig. 1 Pepal: protocol's high-level view: The sender transfers a watermarked version of the document to the receiver. The watermark is the secret key of the receiver unknown to the sender. Upon unauthorized sharing publicly, the sender extracts the watermark (secret key) and penalizes the receiver by transferring the pre-setup deposit

- Sender Privacy: Before the transfer completes, no information regarding the document is available to the receiver.
- Receiver Privacy: Before the disclosure of the document by the receiver, no information regarding the receiver's signing key is available to the sender.
- Sender Integrity: In case of false accusation by the sender, no action is taken.
- Receiver Integrity (Revealing property): In case of disclosure of the document by the receiver, the signing key of the receiver is revealed to the sender.

We formalize these properties as an ideal functionality in Fig. 8 in Sect. 7.

*Solution Overview*We propose the Pepal: protocol, depicted in Fig. 1 involving the two parties, *Sender* and *Receiver*. The sender has the multimedia document M, and the receiver has the signing key sk. The receiver initially makes a time-locked bitcoin deposit of an agreed value of funds that can be opened only if the signing key of the receiver is available. The sender divides the document into several blocks and creates two watermarked versions (corresponding to 0 and 1) for *each block*. The parties run multiple *1-out-of-2* Oblivious Transfer (OT_1^2) protocol instances, one for the transfer of each of the document blocks. The sender uses the watermarked blocks as inputs while the receiver uses each of the bits of his signing key as choice bits for the OT_1^2 s and obtains one version of each block , i.e., for a 256-bit signing key of the receiver, the sender (in the simplest case) divides the document into 256 blocks and creates two versions for each block using *robust watermarking*. The sender and receiver then perform 256 OT_1^2 s, where the choice bit for each OT_1^2 is each of the bits of the 256-bit key of the receiver. The receiver also proves to the sender in zero-knowledge that the signing key used for the deposit is indeed formed of the bits used for OT_1^2 s.

As the document is transferred through oblivious transfer, the sender can not gain any information about the sign-ing key of the receiver. However, suppose the document is revealed/disclosed before the time of the expiry of the agreement. In that case, the sender learns the signing key of the receiver from the watermark of the revealed document. He can then penalize the malicious receiver by transferring the funds to himself. The multiple OT_1^2 s, one for each block, ensure that the watermark embedded in the document corresponds to the signing-key bits.

To transfer the funds out of the deposit, the sender needs both his and the receiver's signature, which can not be obtained before the document is revealed to the public. Thus, he can penalize the receiver only if she is dishonest. If the receiver is honest, the agreement will expire after the agreed time, and the funds will be transferred back to her. The transactional logic of the deposit is depicted as a pseudo-code in Algorithm 1.

Algorithm 1 Claim-or-Refund contract logic

```
if Current time t_now ≥ t then
    Direct the locked funds back to the contract creator
else
    if Both the sender and receiver sign the transaction then
        Direct the funds to the mentioned recipient
    else
        Transaction is invalid
```

The receiver, instead of full disclosure, can disclose the document partially to the public. She can reveal, say, half of the total 256 blocks received so that only half the number of bits of her signing key are revealed to the sender. However, for a 256-bit key of the receiver, the sender can in fact divide the document into more numbers of blocks than just 256. This way, he can embed the key multiple times in the document; for example, the sender can divide it into 512 parts so that the key gets embedded twice. The sender can perform 512 OT_1^2 s with the receiver using her 256-bit key twice for the same. In such a scenario, the sender can extract more number of bits upon partial disclosure. Also, the information in the document may not be "uniform" throughout the document, so the sender can also try to embed the key multiple times in a document part where there is "more" information by dividing it into more parts at those document locations.

The receiver understands that one bit of her signing key is watermarked in each document block received using that bit in OT^2_1. She also knows which particular bit is embedded in a particular document block; this is because, the watermark embedded in a block is the same as the choice bit used for OT^2_1 in obtaining a document block. Leveraging this knowledge, the receiver can try to minimize the number of bits revealed to the sender. For example, with the sender dividing the document into 512 blocks and the receiver having a 256-bit signing key, the receiver can reveal 100 blocks of the received document revealing only 50 bits to the sender. She can achieve this by revealing two blocks received with each bit for 50 bits. To prevent such an attack, we propose a primitive called *Doubly Oblivious Transfer* (DOT). DOT prevents the receiver from learning which bit (index of the bit) of her key is watermarked into a particular block.

In DOT, the sender has two messages m_0, m_1, and the receiver has two bits s_0, s_1 (refer Fig. 2 in Appendix). The sender has an extra choice bit c, using which he transfers m_{s_c} (associated with the bit s_c) to the receiver. At the end of DOT instance, the receiver cannot determine the value of c and m_{1-s_c} and the sender does not know the bit s_c that has been used in the transfer of m_{s_c}. Refer to Fig. 2 in the Appendix for the pictorial depiction of the simplest form of DOT protocol.

For Pepal:, the sender can use DOT to transfer the document to the receiver such that she has no information about which of her bits is embedded in a particular document block. As we analyze in Sect. 7.1, this greatly improves the expected number of bits revealed to the sender in case of partial disclosure. For example, with the sender dividing the document into 512 blocks and 256-bit key at the receiver, upon disclosure of 100 blocks, the expected number of bits that the sender can extract is 90.3 instead of 50 while using just oblivious transfer.

Notice that our Pepal: protocol augments cryptographic primitive with a smart contract. Given the limited expressibility of Bitcoin contracts, our (off- chain) cryptographic solution seems necessary, but this may not be the case for Turing complete systems like Ethereum [5]. However, defining the complete solution as a smart contract will not be or may not remain inexpensive enough. Further comments regarding the contracts can be found in Sect. 9 .

3 Building blocks and cryptographic tools

In this section, we introduce the building blocks and cryptographic tools required for the protocol – robust watermarking, oblivious transfer and claim-or-refund contract.

[1] For $s_0 = s_1 = b$, the receiver knows that she received m_b; however, that does not constitute any privacy leakage in our application as c and m_{1-s_c} remain private.

Robust Bit Watermarking A robust watermarking scheme is defined by the property that the watermark can not be removed without loss of information from the watermarked data. The watermarking scheme is defined by three algorithms, for key generation, embedding the watermark and detection of the watermark. M is the set of all possible documents, $W \subseteq \{0,1\}$ the set of all possible watermarks, K is the set of all keys and κ is the security parameter. The three algorithms define the scheme:

wm.gen (κ): Given κ, outputs keys $k_{emd}, k_{det} \in K$ probabilistically.

wm.embed (M, w, k_{emd}): Takes the document M, watermark $w \in W$ and embedding key k_{emd} as inputs and generates a watermarked document M'.

wm.detect (M', k_{det}, w): Takes the watermarked document M', the detection key k_{det} and the watermark w as input and outputs $\top$ if the watermark in M' matches w, else outputs $\bot$.

The watermarking scheme is expected to satisfy the properties of imperceptibility and robustness. To describe the properties, we adapt the watermarking definition suggested by Adelsbach et al. [9]. We assume a given similarity function $sim(M, M')$ which returns $\top$ if the two documents M and M' are not similar and $\bot$ if they are.

- *Imperceptibility*: The watermarked and the original versions of the document should be similar i.e., $\forall M \in \mathcal{M}, \forall k_{emd} \in K$ and $\forall w \in W$, if $\mathsf{wm.embed}(M, w, k_{emd}) \rightarrow M'$, then $sim(M, M') = \top$.
- *Robustness:* No known algorithm should be able to effectively change or remove the watermark in the watermarked document without leaving the document itself unusable, even with the detection key.

The Pepal: protocol uses a robust watermarking scheme to watermark either the bit 0 or the bit 1. The actual watermarking scheme varies depending on the type of the data being watermarked. While theoretically, an algorithm may exist which can remove the watermark from the data, we just require that such an algorithm should not be available or known to humans; this approach was formalized by Rogaway[42]. We discuss the robust watermarking algorithms in Sect. 8.

Oblivious Transfer 1-out-of-2 oblivious transfer (OT^2_1) is a two-party (a sender and a receiver) computation mechanism, where the sender has two messages M_0 and M_1 and the receiver has a bit $b \in \{0,1\}$. The goal is to transfer M_b to the receiver and at the end of the protocol, the receiver should not learn any information about M_{1-b} and the sender should not learn b. We consider the oblivious transfer protocol, called the Verified Simplest OT by Doerner et al. [22] which is an

extended version of OT protocol by Chou et.al. [19], recalled in Appendix A along with Fig. 11.

The multiplicative group G used for the protocol is Gap-DH [27] and the additional verification step forces the receiver to make oracle queries before receiving the encryptions from the sender, there by making the protocol UC-Secure.

Bitcoin Claim-or-Refund Contract Bitcoin [40] is a peer-to-peer decentralized network where participants are represented by a public and private key pair. The hash of the public key serves as the user's address and the private key is used to sign and authorize transactions. *Script* in Bitcoin is a stack-based language simulating a Push Down Automata and is used to write a smart contract. Spending funds typically involves executing/running two scripts on the spender's machine. The first is scriptPubKey which is embedded in the input transaction under the script field. It entails the conditions that must be met to spend the unspent transaction outputs (UTXO). The second one is *scriptSig* which is an unlocking script provided by the user who wants to spend the UTXO. When *scriptSig* and *scriptPubKey* are executed in sequence, the user gets to know if the transaction is valid. Bitcoin offers both sender and receiver of the funds an aspect of privacy until the funds in the deposit are directed to a recipient i.e., in our case, after the documents become public and the key gets revealed to the sender. Such privacy is not observable in any other non-blockchain financial system.

Time-Locked Compensation Deposits: We construct scriptPubKey with two prominent Bitcoin scripting language operators: OP_CHECKLOCKTIMEVERIFY and OP_CHECKMULTISIGVERIFY.

OP_CHECKLOCKTIMEVERIFY allows users to create transactions whose outputs can only be spent in the future. OP_MULTISIGVERIFY allows the creation of transactions which need multiple signatures. In our case, the receiver creates a deposit which is locked till a future time t. The funds of the deposit can be transferred only if both the signatures of sender and the receiver are submitted before the time t. After time t, the unspent funds are transferred back to the receiver. Embedding such instructions into the funds is commonly referred to as a smart contract. Our smart contract automates the claim-or-refund functionality. The funds are transferred either when the time of the agreement expires or when the signatures of both sender and receiver are available.

The scriptPubKey that receiver uses in the contract is

```
IF
OP_CHECKLOCKTIMEVERIFY OP_DROP
 pkR  OP_CHECKSIGVERIFY
ELSE
OP_2  pkR pkS  OP_2
      OP_CHECKMULTISIGVERIFY
ENDIF
```

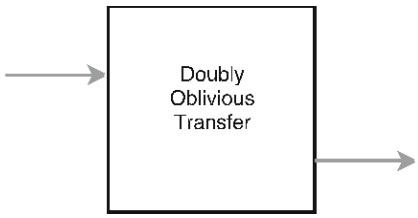


Fig. 2 Doubly Oblivious Transfer Primitive: the sender has two input messages m_0, m_1 and a bit c. The receiver has two input bits s_0, s_1 and obtains m_{s_c}

4 Doubly oblivious transfer — DOT

In our solution, the receiver obtains the document blocks by running OT_1^2 multiple times with her signing key bits as the choice bits. However, while running OT_1^2, the receiver understands that each of the message that is received by using choice bit is indeed affected by the choice bit i.e., the receiver knows the index of the bit embedded through watermark in a received message/document block.

To overcome this, we propose a primitive, in which the receiver, after giving multiple bits as input, receives several messages corresponding to the input bits, but the receiver does not have any information about which bit was used as choice bit for choosing a certain message. In the simplest case the sender has two messages m_0, m_1 along with a choice bit c and the receiver has two bits s_0, s_1 as depicted in Fig. 2. The sender chooses one of the indices of the bits of the receiver using the bit c and the receiver receives the message m_{s_c} corresponding to the bit of the chosen index. Here, the sender does not know which message has been received by the receiver and the receiver does not know which of her two bits is chosen as the choice bit to choose the messages. Hence we call it *Doubly Oblivious Transfer* (DOT) protocol.

Figure 3 represents the ideal functionality of the DOT protocol. The functionality $\mathcal{F}_{\mathsf{DOT}}$ interacts with the sender S and receiver R. The adversary $\mathcal{A}$ controls the communication and the delivery of the messages. When the sender and receiver forward their inputs, they use the tags `inputS, inputR` respectively. The functionality delivers the corresponding messages to sender and receiver on receiving the messages `deliverS, deliverR` from the adversary. DOT hides the index c and m_{1-s_c} from the receiver, but it need not essentially hide the value s_c itself. For $s_0 = s_1 = b$, the receiver knows the value b but not c.

Each session of the protocol run is identified by a session id sid. The sender S forwards the two messages M_0, M_1 and the choice bit c along with the tag `inputS` to the functionality indicating the input from sender S. Upon the initiation of the session by the sender, the functionality informs the receiver R by forwarding the `intd` message along with the session id sid. The session initiation is also intimated to the

Functionality F_{DOT}

Ideal functionality F_{DOT} interacts with sender S and receiver R. The sender has two messages M_0, M_1 and a choice bit c. The receiver has two bits s_o, s_1. The adversary A corrupts either the sender or receiver.

Sender input. Upon receiving the message (inputS, M_0, M_1, c, sid) with $M_0, M_1 \in \{0,1\}^{\square}$, $c \in \{0,1\}$ from sender S, record S, M_0, M_1, c, sid, forward the message (intd, sid) to the receiver R and (input, sid) to A.

Receiver input. Upon receiving the message (inputR, s_0, s_1, sid) with $s_0, s_1 \in \{0,1\}$ from receiver R, record R, s_0, s_1, sid, forward the message (input, sid) to A.

Sender output. Upon receiving the message (deliverS, sid) from A, check if R, s_0, s_1, sid is stored, else ignore the message. Send (delivered, sid) to S.

Receiver output. Upon receiving the message (deliverR, sid) from A, check if S, M_0, M_1, c, sid is stored, else ignore the message. Forward (output, M_{s_c}, sid) to R.

Fig. 3 Ideal functionality of DOT

adversary A. Upon the initiation of the session, the receiver forwards the two bits s_0, s_1 to the functionality along with the session id to the functionality. The adversary controls the delivery of messages from the functionality, this is modelled by the `deliverS, deliverR` messages from the adversary. On receiving the `deliverS` message with the session id sid, the functionality checks if the a corresponding previous input from the receiver is received, if yes, the sender is intimated that the receiver input has been received by forwarding the `delivered` message to the sender. Finally upon receiving the `deliverR` message along with sid from the adversary, the functionality checks if it received the input from the sender for that session id and if yes forwards the message M_{s_c} to the receiver.

Construction We provide a construction which realizes the ideal functionality of DOT with two messages M_0, M_1 and a choice bit c at the sender and two bits s_0, s_1 at the receiver as given in the Fig. 3. Both the parties possess public key-secret key pairs (refer Fig. 4) and $pk = pk_S \| pk_R$ where pk_S, pk_R are public keys of sender and receiver. The sender samples two elements from the group (can be points from the elliptic curve), encrypts the two messages using a symmetric encryption $E_{(.)}(.)$ with the keys obtained by hashing the elements. These encryptions are randomly permuted and forwarded to the receiver in the form of Enc_i. This is the first step in DOT. The sender then transfers the elements to the receiver such that the receiver can only decrypt M_{s_c}. The

Fig. 4 Doubly oblivious transfer (DOT) protocol

Sender		**Receiver**
Message blocks: M_0 and M_1, Choice bit: c		Bits: s_0, s_1
	Setup	
	for permutation	
	El-Gamal encryption of group elements	
Set $g_{c,0} = g_0, g_{c,1} = g_1$		
$g_1{}_{c,0}; g_1{}_{c,1} \leftarrow_R \mathbb{G}$		
$u_{i,0} = \mathcal{E}_{pk}(g_{i,0}), u_{i,1} = \mathcal{E}_{pk}(g_{i,1})$		
	Oblivious Transfer [22]	
	Run OT_1^2 for i	
Input $u_{i,0}, u_{i,1}$		Input s_i
		Output u_{i,s_i}
	Re-randomization, Forwarding and Decryption	
		$v_{i,s_i} = \mathcal{R}_{pk}(u_{i,s_i})$
	$\xleftarrow{v_{i,s_i}}$	
$x_{i,s_i} = \mathcal{D}_{sk_S}(v_{i,s_i})$	$\xrightarrow{x_{c,s_c}}$	
		$g_{c,s_c} = \mathcal{D}_{sk_R}(x_{c,s_c})$
		Decrypt $\widehat{Enc}_{s_c}$ using $H(g_{c,s_c})$ appropriately

encryption and forwarding of messages prevents the need to map random message strings onto group elements for the ElGamal encryption in the next step.

The sender samples two more elements, populates $g_{i,j}$, $i, j \in \{0, 1\}$ as shown in Fig. 4 and encrypts all $g_{i,j}$ to the public key pk using $E_{pk}(.)$ - a Re-randomizable encryption like ElGamal encryption to obtain $u_{i,j}$. Now two OT_1^2 instances are run, one for each i with $u_{i,j}$ as inputs. The receiver inputs s_i as the choice bit for the instance i of OT_1^2.

Here the sender S initiates the protocol, this would correspond to the `inputS` message of the functionality. Once the protocol is initiated, the receiver inputs the bits s_0, s_1 into the OT_1^2 protocol instance. This would correspond to the `inputR` message of the ideal functionality messages where the receiver forwards the bits s_0, s_1.

The encryption of the elements to the key pk later helps the receiver to hide which keys have been obtained by her through OT_1^2 and helps the sender to hide the order in which the keys have been forwarded. Hiding the order implies hiding the mapping between bits s_i and elements obtained by the receiver through OT_1^2. The receiver after receiving the different u_{i,s_i} through OT_1^2 proceeds by applying $R_{pk}(.)$, a re-randomization operation to obtain v_{i,s_i}. These re-randomized encryptions of obtained encrypted elements are now forwarded back to the sender. If there was no re-randomization step, the sender would know what elements have been obtained by the receiver and so will know what version of the message was taken by the receiver. Hence we use the re-randomization step to hide from the sender, information regarding which messages have been obtained by the receiver through OT_1^2. The sender from v_{i,s_i}, decrypts his layer of ElGamal encryption using the decryption operation $D_{sk_S}(.)$ to obtain x_{i,s_i}. He then drops x_{1-c,s_i} and forwards only the element x_{c,s_c} to the receiver. The element x_{c,s_c} (which at this point is only encrypted to the receiver's public key) is then decrypted by the receiver using her private key using $D_{sk_R}(.)$ to obtain the element g_{c,s_c}. The key obtained as hash of g_{c,s_c} is used to decrypt the initially obtained random permutation of messages. Only one of them gets decrypted *correctly*. The receiver, while decrypting the encrypted messages, would not know which message is the correct encryption using the obtained key, she tries to decrypt each of the messages. For the receiver to be able to recognize the correct message for the key, we need a mechanism.

To achieve the decryption and identification of the correct message block by the receiver, the sender initially appends each of the messages with a string which is obtained as a certain public function $f(.)$ of key (like hash of the key) used to encrypt the message before the encryption process. After decrypting each block with the key, the receiver matches the appended string with the locally calculated string using $f(.)$ of the key. Whichever message has the correct match, is the correct message. Thus the receiver decrypts M_{s_c}. Here it can be seen that only the receiver obtains an output from the protocol, this is modelled through the `output` message in the ideal functionality.

Imagine the case when the initial encryptions are not permuted, then the receiver knows that the the encryptions received correspond to bit indices 0 and 1 in that order, so she can try to attack the system by setting one of the bits, say $s_0 = 0$ and the other $s_1 = 1$, then which ever encryption gets decrypted, will reveal which of the two s_is has been chosen by the sender. To prevent such a scenario, the initial permutation of the encryptions is necessary.

Universal Composability [13] Let $\mathsf{EXEC}_{\rho,A,E}$ be the ensemble of the output of the environment E when interacting with the adversary A and parties running the protocol ρ over the random coins of all the involved machines

Definition 1 A protocol ρ UC-realizes an ideal functionality F if for any adversary A, there exists a simulator such that for any environment E, the ensemble $\mathsf{EXEC}_{\rho,A,E}$ and $\mathsf{EXEC}_{F,S,E}$ are computationally indistinguishable.

Theorem 1 *The DOT protocol UC-realizes the functionality F_{DOT} in the F_{OT}-hybrid model under the following conditions:*

Corruption Model: *Static corruption (the sender or receiver is corrupted at the beginning of the protocol).*

Hybrid Functionalities: *H is modelled as a random oracle and secure channels between the parties are assumed.*

Computational Assumption: *The encryption scheme used in the initial step is symmetric, non-committing and robust [19]. Group used for OT_1^2 module* G *is a Gap-DH group.*

We provide proofs of all the theorems in 1.

5 Generalization of DOT protocol

The DOT protocol can be easily extended to work with multiple messages at the sender and κ-bit signing key of the receiver as shown in Fig. 5.

In the general case, the sender has a total of 2κ messages $M_{i,j}$, for $0 \leq i \leq \kappa - 1$, $j \in \{0, 1\}$ and the receiver has bits s_n, $0 \leq n \leq \kappa - 1$. After participating in the protocol, the receiver receives $M_{i,l}$, $l = s_{\pi(i)}$ for a permutation π of set of indices i chosen at the sender. The permutation of indices is the general case equivalent of the choice bit c of the two bit case.

Forwarding a random permuted order of encrypted messages remains similar for the general case. When the elements are sampled in the general case, sampling extra elements is not *necessary*. The sender performs a permutation π on the rows i of the elements $g_{i,j}$ to obtain $g_{i,j}$ which are encrypted using $E_{pk}(.)$ as before. Now, $g_{i,j}$ are input to i instances of OT_1^2 to which the receiver inputs s_i as the choice bits for

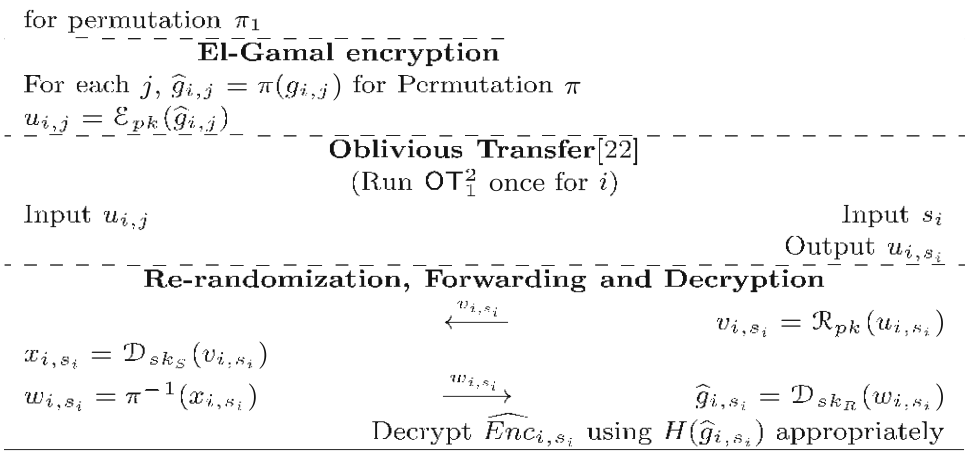


Fig. 5 Doubly oblivious transfer protocol (General Case)

each instance i. The receiver obtains u_{i,s_i}, re-randomizes the encryption using $R_{pk}(.)$ and sends back v_{i,s_i}. After receiving v_{i,s_i}, the sender reverses the permutation order to obtain $w_{i,s_i} = \pi^{-1}(v_{i,s_i})$. He then decrypts his layer of encryption using $D_{sk_S}(.)$ and forwards x_{i,s_i} to the receiver who decrypts her layer of decryption to obtain g_{i,s_i}. These g_{i,s_i} are hashed to obtain the final keys which are then used to decrypt the Enc_i received in the first step. Note that if the number of messages is not a multiple of 2κ, the sender can sample extra elements and encrypt them to input them in OT_1^2. After receiving the encrypted elements from the receiver, he can discard the elements at the indices where the extra elements have been placed in the OT_1^2 step. Also, if the receiver tries to attack the protocol by manipulating the cipher texts after the re-randomization step, she will not be able to receive meaningful keys for the correct decryption, she can gain no information regarding the sender's messages or permutation applied on encrypted messages.

6 Committed receiver oblivious transfer

An oblivious Transfer instance transfers one message M_b where $b \in \{0, 1\}$ of the two messages M_0 and M_1 from the sender to the receiver with bit b. In our protocol which uses DOT (which in-turn uses OT_1^2), we further *require* the bit b to be a bit of the signing-key of the receiver. With a simple OT_1^2, the sender can not be sure if that is the case. To overcome this, we propose the committed receiver oblivious transfer (CROT) primitive.

In CROT, the receiver forwards a non-interactive zero knowledge (NIZK) proof of knowledge to prove that the bit inputs from the receiver are in fact bits of the signing key. The functionality of the protocol CROT is presented in the Fig. 6. The functionality F_{CROT} interacts with the sender S and receiver R. The sender has 2κ messages $M_{i,j}, i \in [0, \kappa - 1], j \in 0, 1$. The receiver has bits s_i which form the secret key sk of the key pair (sk, pk). The adversary A con-trols the communication and the delivery of the messages.

The sender S uses the session id sid and tag `inputS` to forward the messages $M_{i,j}$ to the functionality and to initiate the protocol instance. The functionality stores the sender input messages using the record S, $M_{i,j}, sid$. The functionality intimates the initiation of the session to the adversary and the receiver using the messages `input, intd` respectively. Upon the intimation, the receiver R forwards the bits s_i with the `inputR` tag to the functionality. After receiving the message (`inputR`, s_i, sid), the functionality stores the record (R, s_i, sid) and intimates the adversary that the input has been received. The adversary sends the

Functionality $\mathcal{F}_{\mathsf{CROT}}$

Ideal functionality $\mathcal{F}_{\mathsf{CROT}}$ interacts with sender S and receiver R. The sender has messages $M_{i,0}, M_{i,1}$ and the receiver has bits $s_i, i \in [0, \kappa-1]$. s_i correspond to the secret key sk whose public key is pk. The adversary $\mathcal{A}$ corrupts either the sender or receiver.

Sender input. Upon receiving the message $(\mathtt{inputS}, M_{i,j}, sid)$ with $M_{i,j} \in \{0,1\}^*$, $i \in [0, \kappa-1], j \in \{0,1\}$ from sender S, record $\langle \mathsf{S}, M_{i,j}, sid\rangle$, forward the message $(\mathtt{intd}, sid)$ to the receiver R and $(\mathtt{input}, sid)$ to $\mathcal{A}$.

Receiver input. Upon receiving the message $(\mathtt{inputR}, s_i, sid)$ with $s_i \in \{0,1\}$, $i \in [0, \kappa-1]$ from receiver R, record $\langle \mathsf{R}, s_i, sid\rangle$, forward the message $(\mathtt{input}, sid)$ to $\mathcal{A}$.

Sender output. Upon receiving the message $(\mathtt{deliverS}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{R}, s_i, sid\rangle$ is stored, else ignore the message. Send $(\mathtt{delivered}, sid)$ to S.

Receiver output. Upon receiving the message $(\mathtt{deliverR}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{S}, M_{i,j}, sid\rangle$ is stored, else ignore the message. Check if the secret key sk formed from the bits s_i forwarded by the receiver corresponds to the public key pk. Else, ignore the message. Forward $(\mathtt{output}, M_{i,s_i}, sid), i \in [0, \kappa-1]$ to R.

Fig. 6 Ideal functionality of CROT

messages deliverS and deliverR to ask the functionality to deliver the outputs to the sender and the receiver. Upon receiving deliverS, the functionality checks if a record R, s_i, sid is stored and if yes, it sends the message $(\mathtt{delivered}, sid)$ to the sender. On receiving the message $(\mathtt{deliverR}, \mathtt{sid})$, the functionality checks if there is a corresponding record $\mathsf{S}, M_{i,j}, sid$. If it exists, it verifies whether the bits s_i forwarded by the receiver correspond to the secret key (sk) of the public key pk. On successful verification, it forwards the messages M_{i,s_i} to the receiver.

We depict the construction of the protocol in Fig. 7 .

Construction The protocol construction for the ideal functionality F_{CROT} as given is the Fig. 6 is presented here. The sender has messages $M_{i,j}$ for $0 \le i \le \kappa - 1$ and $j \in \{0, 1\}$. The receiver has a signing key sk (s_i for $0 \le i \le \kappa - 1$ are the bits of sk). The sender and receiver inputs are modelled using the inputS and inputR messages of the function-ality; the sender initiates the protocol using the inputS message. Given a multiplicative group G and its generator g, the sender initially chooses a random value $a \leftarrow \mathsf{Z}_q$ and forwards $\mathsf{h} = \mathsf{g}^a$ to the receiver. This would be the *Setup* phase. In the next *Commit and Prove* phase, the receiver chooses random $r_i \leftarrow_R Z_q$ and computes $\mathsf{c}_i = \mathsf{g}^{r_i}\mathsf{h}^{s_i}$ for $0 \le i \le \kappa - 1$. The c_i values are forwarded to the sender as commitments to the bits s_i. The receiver also forwards $r = \sum_{i=0}^{\kappa-1} 2^i r_i$ to the sender. Along with these, for $0 \le i \le \kappa - 1$, the receiver forwards non-interactive zero knowledge (NIZK) proofs of knowledge of exponents [17] r_i and s_i such that $\mathsf{c}_i = \mathsf{g}^{r_i + a s_i}$.

Each of these NIZK proofs is realized using the standard Fiat-Shamir transformation [25] of an interactive sigma protocol for Pedersen commitments in the random oracle model. Following the formal symbolic notation introduced by Camenisch and Stadler [17], each proof is depicted as $\mathsf{PoK}\{(r_i, s_i) | \mathsf{g}^{r_i}\mathsf{h}^{s_i}\}$ in Fig. 7. This phase is used by the receiver to prove that the bits s_i used for the transfer are indeed the bits of the signing key sk. The sender verifies if $\mathsf{c} = \mathsf{g}^r \mathsf{pk}^a$ for the computed $\mathsf{c} = \prod_{i=0}^{\kappa-1} c_i^{(2^i)}$. He also verifies the NIZK proof. If both the verifications succeed, he proceeds with the protocol, else, aborts. The verification would also fail if (pk, sk) are not a key pair. This verification corresponds to the verification in the 'Receiver output' step of the functionality of Fig. 6.

After successful verification the sender computes the keys $k_{i,j} = H((\mathsf{c}_i \cdot h^{-j})^a)$ for each $0 \le i \le \kappa - 1$ and $j \in \{0, 1\}$. The sender verifies if the receiver computed the keys using the verification step similar to Verified Simplest OT [22]. He forwards the challenges $p_i = H(H(k_{i,0})) \oplus H(H(k_{i,1}))$ for each i and receives the responses in the form of p_i and the sender verifies if $p_i = H(H(k_{i,0}))$. The keys $k_{i,j}$ are used to encrypt messages $M_{i,j}$ respectively to obtain the cipher texts $C_{i,j}$. The cipher texts $C_{i,j}$ are forwarded to the receiver who attempts to decrypt the blocks C_{i,s_i} using the keys k_{i,s_i} finishing the *Transfer* phase. The receiver can not compute the keys $k_{i,1-s_i}$ (follows from Lemma 1 of [19]) and so can not decrypt $C_{i,1-s_i}$. It can be observed that at the end of the protocol only the receiver obtains an output in the form of the messages M_{i,s_i}, this is modelled throught the output message in the 'Receiver output' step of the functionality. One can observe that the protocol does not enforce the receiver to use "bits", if the receiver uses any other values other than bits in CROT, the receiver receives encryptions which can not be decrypted.

The model for CROT includes static corruption of parties, modelling H as random oracle and group G being Gap-DH [27] while the encryption used is symmetric, non-committing and robust [19].

Theorem 2 *The CROT protocol UC-realizes the ideal functionality F_{CROT} in the F_{ZK}^{DL}-hybrid model under the following assumptions:*

- ***Corruption Model:*** *static corruption*
- ***Hybrid Functionalities:*** *H is modeled as a random oracle and authenticated channels between users are assumed.*

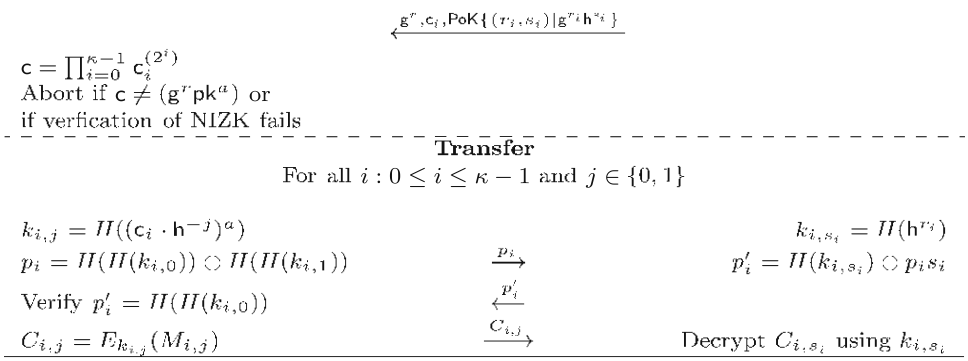


Fig. 7 Committed receiver oblivious transfer (CROT) Protocol

- ***Computational Assumptions:*** G *is Gap-DH. The symmetric encryption used is non-committing and robust.*

7 The Pepal: protocol

Here, we detail the steps of the Pepal: protocol which uses DOT with CROT. The watermarking and the DOT protocol are the *off-chain* cryptographic components while the smart-contract and the deposit are the *on-chain* parts.

1. **NetworkSetup**: The sender and receiver setup their Bitcoin identities by generating secret key-public key pairs; the sender has the document M.
2. **DepositSetup***(sk, t, Value)*: A time-locked bitcoin deposit is created by the receiver with the signing key sk for a time t and for a amount of $Value$. The deposit is a 2-of-2 multisig deposit requiring the secret keys of both the sender and the receiver to transfer the funds.
3. **WaterMark***(M)*: The document M is broken into κ blocks $M_i, 0 \leq i \leq \kappa - 1$ for a κ-bit long sk and each block M_i is watermarked to generate two versions $M_{i,0}$, $M_{i,1}$. Any watermarking scheme which satisfies the previously mentioned properties (refer Sect. 3) can be used.
4. **DOT with CROT***($M_{i,0}$, $M_{i,1}$, sk)*: The Doubly Oblivious Transfer protocol, used to transfer the document, takes the watermarked blocks as input. In Pepal:, the DOT protocol instead of using OT_1^2, uses CROT. The protocol is same as the general case of DOT (as shown in Fig. 5 of Appendix) but uses CROT instead of OT_1^2. The sender watermarks the document blocks to obtain $M_{i,j}$, generates keys from sampled group elements and forwards the permuted symmetric encrypted versions of the blocks to the receiver. He then encrypts the group elements using El-Gamal encryption to the key $pk = pk_S \boxdot pk_R$ where pk_S, pk_R are the public keys of sender and receiver. The sender inputs encrypted elements in a permuted order to the CROT protocol. The receiver after proving in zero knowledge that the input to the protocol is her signing key sk, receives a set of encrypted elements which she re-randomizes and sends back. The sender, decrypts his layer of encryption, inverts the applied permutation to obtain the elements in their original order and forwards them to the receiver who will be able to decrypt them. The appropriately decrypted symmetrically encrypted blocks are then joined together to form the receiver's version of the document M_{sk}.
5. **Penalize***(M_{sk}, sk_S)*: Upon revelation of the document, the receiver's secret key sk is extracted from the document M_{sk} and is used with the sender's secret key sk_S to transfer the deposited funds to the sender to penalize the receiver.

Utilizing Bitcoin Before the Pepal: protocol begins, after the two parties agree on the Pepal: process, the sender shares his/her public key pk_S with the receiver to create a deposit. The sender will assert that the receiver creates a transaction TX that is valid for a mutually agreed upon time t, and can be redeemed by the sender instantly with the signing keys of the

sender (sk_S) and the receiver(sk_R). Here, the deposit should hold the funds equal to an agreed upon value $Value$. *VerifyDeposit*(TX) at the sender verifies the above mentioned criterion. This algorithm receives the hash of the transaction as an input and verifies that the transaction meets the above mentioned criteria, i.e. it is a valid deposit that directs $Value$ to the sender if the sender has both the signing/private keys. Earlier versions of Bitcoin allowed senders to broadcast time locked transactions and these transactions would be in the unverified transactions pool until the time lock expired or an unlocking *scriptSig* was provided by the spender of TX. However, current (as of February 2019) Bitcoin transaction does not permit nodes to propagate transactions that have an active time lock. Therefore, the receiver sends TX over any secure communication channel so that the sender can verify and sign the transaction. Once the document becomes public, we are assured from the watermarking scheme that the leaked copy of the document will have the receiver's signing key. Using the extraction algorithm *Extract*(M, M_{sk}) the sender can reconstruct the signing key sk. Once the sender has sk, he can sign the transaction TX with the *Sign*(TX, sk) and broadcast the signed transaction directing the funds in TX to his Bitcoin address.

Note that, if the sender wishes to release the data without any penalization (before the condition is met), he can forward the (partial) signature using his secret key to the receiver who can use it along with signature with own secret key to obtain the deposit.

*Ideal functionality*Fig. 8 presents the ideal functionality $F_{\mathsf{Pepal:}}$ for Pepal:, while Theorem 3 proves its security.

The functionality $F_{\mathsf{Pepal:}}$ interacts with the sender S and receiver R. The sender has 2κ watermarked messages $M_{i,j}$, i ∈ [0, κ − 1], j ∈ 0, 1. The receiver has bits s_i which form the secret key sk of the key pair (sk, pk). While forwarding the input the sender also forwards a permutation $\pi(\cdot)$ of indices [0, κ − 1]. Before delivering the messages $M_{i,l}$, l = $s_{\pi(i)}$ to the receiver, the functionality checks if the input bits s_i form the secret key sk corresponding to the public key pk.

Here we show that the functionality achieves the desirable properties discussed in Sect. 2. The properties of sender and receiver privacy are trivially satisfied by the functionality as it does not reveal any information except transferring the corresponding watermarked blocks to the receiver. If the receiver discloses the document, the sender can extract the embedded watermark bits and hence the signing key of the receiver, thus satisfying the revealing property. If the sender tries to falsely accuse the receiver by revealing the document in any form, the receiver does not lose the deposit as the sender does not have the receiver's key without disclosure, this achieves the sender integrity property. Though the penalization is shown as a step of Pepal:, as it takes place outside of the transfer mechanism after the data breach in a non-interactive way, it is not included in the ideal functionality of the Pepal: protocol.

Functionality $\mathcal{F}_{\mathsf{Pepal}}$

Ideal functionality $\mathcal{F}_{\mathsf{Pepal}}$ interacts with sender S and receiver R. The sender has watermarked messages $M_{i,0}, M_{i,1}$ and the receiver has bits $s_i, i \in [0, \kappa - 1]$. s_i correspond to the secret key sk where public key is pk. The adversary $\mathcal{A}$ corrupts either the sender or receiver.

- Upon receiving the message $(\texttt{inputS}, M_{i,j}, \pi(\cdot), sid)$ with watermarked $M_{i,j} \in \{0,1\}^*$, $i \in [0, \kappa - 1], j \in \{0,1\}$, a random permutation $\pi(\cdot)$ from sender S, record $\langle \mathsf{S}, M_{i,j}, \pi(\cdot), sid \rangle$, forward the message $(\texttt{intd}, sid)$ to the receiver R and $(\texttt{input}, sid)$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{inputR}, s_i, sid)$ with $s_i \in \{0,1\}$, $i \in [0, \kappa - 1]$ from receiver R, record $\langle \mathsf{R}, s_i, sid \rangle$, forward the message $(\texttt{input}, sid)$ to $\mathcal{A}$.
- Upon receiving the message $(\texttt{deliverS}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{R}, s_i, sid \rangle$ is stored, else ignore the message. Send $(\texttt{delivered}, sid)$ to S.
- Upon receiving the message $(\texttt{deliverR}, sid)$ from $\mathcal{A}$, check if $\langle \mathsf{S}, M_{i,j}, sid \rangle$ is stored, else ignore the message. Check if the secret key sk formed from the bits s_i forwarded by the receiver corresponds to the public key pk. Else, ignore the message. Forward $(\texttt{output}, M_{i,l}, sid), l = s_{\pi(i)}, i \in [0, \kappa - 1]$ to R.

Fig. 8 Ideal functionality of Pepal:

Theorem 3 *The Pepal: protocol securely implements the ideal functionality $F_{\mathsf{Pepal:}}$ in the F_{DOT}, F_{CROT} hybrid model under the following assumptions:*
Corruption Model: static corruption
Hybrid Functionalities: *H is modeled as a random oracle and authenticated channels between users are assumed.*
Computational Assumptions: *CDH and DDH are assumed to be hard in* G, G *is Gap-DH. The symmetric encryption used is non-committing and robust.*

7.1 Illustration

We illustrate the utility of Pepal: with DOT using CROT with an example. The sender can break the document down into more than κ blocks, say 2κ, to perform CROT twice, there by embedding the receiver's key two times. The finer he breaks the document, the more number of times he will be able to embed the receiver's key and so can extract more number of bits upon partial disclosure. For a receiver with 256 bit key, the sender for embedding the key twice divides the document

Table 1 Time (mean ± standard deviation) taken (in seconds) for steps of the protocol when signing key is embedded for = 1, 4 and 16, is the number of copies of secret key embedded in the data

	Watermarking	Full protocol
= 1	0.357 ± 0.009	1.737 ± 0.226
= 4	1.346 ± 0.213	16.067 ± 0.638
= 16	1.643 ± 0.283	83.101 ± 1.623

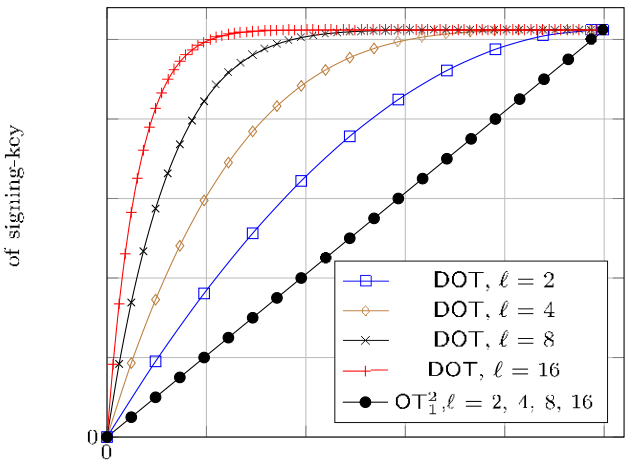


Fig. 9 Number of bits revealed to sender upon dishonest disclosure by receiver when Pepal: is employed with OT^2_1 and DOT with 256-bit signing key. is the number of copies of secret key embedded in the data

into 512 blocks and creates two watermarked versions for each of the 512 blocks and wishes to transfer 512 messages.

The receiver wishes to selectively reveal parts of the document to the public while not revealing too much of her key bits to the sender. It is understood that the receiver reveals at least enough number of blocks (not too few) to carry useful/sufficient information. Let us assume she wishes to reveal 100 document blocks. We wish to compare how many bits she will actually reveal to the sender when she reveals 100 document blocks when Pepal: with DOT is used, to a scenario where just OT^2_1 is used to transfer the messages instead of DOT.

If the sender uses just OT^2_1 for the message transfer, he inputs one pair of messages for each OT^2_1 and performs 512 such OT^2_1 instances to transfer the 512 messages. In this case, the receiver knows which document block has been obtained using a particular key bit and so knows which two blocks have a certain key bit embedded in them. As she knows which two blocks have the same bit embedded in them, she will reveal 50 such pairs (with the same key bit) to the public so that the sender can learn only 50 of her signing key bits .

However, if the sender uses DOT with CROT to transfer the document and the receiver decides to reveal 100 document blocks, as she does not know which key bit is embedded in a certain document block, she randomly picks 100 document blocks and reveals to the public. The expected number of key bits revealed to the sender in such a scenario would be 90.3 for 100 blocks as opposed to 50 bits with just OT^2_1. Following [28, 45], the expected number of bits revealed to the sender when m blocks of the document are released with κ -bit key being watermarked over times in the document is κ

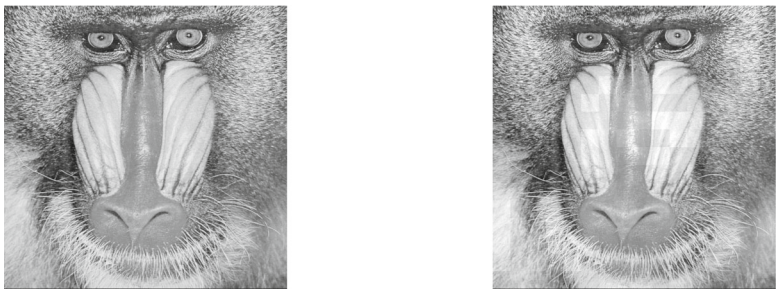
$$1 - \frac{(\kappa-1)^{\square}/\kappa^{\square}}{m \qquad m}.$$

Figure 9 indicates the number of bits revealed to the sender against the percentage of blocks revealed to the public when the signing-key is watermarked times with {2, 4, 8, 16}. When the key is embedded 8 times, a leakage of 20% of the document/file can leak up to 211 bits of the key whereas, when it is embedded 16 times, even a 15% leak reveals as many as 235 bits. This scenario is particularly useful with larger files like video files, where the key can be embedded many number of times such that even a minor clip of the video can reveal close to the whole of the signing key. The plot in the Fig. 9 compares the number of signing-key bits revealed to the sender when Pepal: uses DOT and OT^2_1. It clearly indicates that higher the number of times the key is embedded, higher are the number of bits revealed to the sender upon leakage. However, one has to note that the maximum number of times a key can be embedded by dividing the document depends on the document and its entropy.

Computation and Communication Overhead For the transfer protocol, the number of exponentiations at the sender and receiver is linear in . When DOT uses CROT, the number of exponentiations performed by the sender would be 11κ + and by the receiver would be 7κ. The communication in the DOT protocol involves forwarding two versions of AES encrypted blocks, messages of CROT and forwarding of κ ElGamal encrypted points by the receiver and the sender. In CROT, the sender forwards 2κ ElGamal encrypted elements while the receiver forwards 3κ elements including the proof of knowledge messages.

8 Implementation and analysis

We have implemented the Pepal: protocol as a single-threaded program and analyzed its performance on a MacOS machine with 3.1 GHz Intel Core i7 and 16 GB RAM. Our implementation involves the DOT protocol with robust watermarked images and a claim-or-refund contract as a Bitcoin script. An execution run involves the transfer of an image to the receiver, and we examine the execution times for the different involved modules. The receiver's key is 256-bit long



(b) Watermarked image reconstructed by the receiver

Fig. 10 Original and reconstructed images

and the sender breaks the document into blocks before proceeding with the protocol.

Watermarking The sender, after creating the document blocks, watermarks each block with 0 and 1 to generate two versions. We employ the watermarking system by Meerwald [38] which implements the Cox algorithm [20] of robust watermarking for the image blocks. The Cox algorithm is well-studied and benchmarked against several attacks [11]. In our scheme, we watermark the image document by embedding the key multiple times, Table 1 indicates the watermarking time taken where the 256-bit is embedded for = 1, 4 and 16 indicating embedding once, 4 and 16 times. For = 1, 4 the document in divided into 256 and 1024 blocks respectively which are transferred using the DOT pro-tocol to the receiver who reconstructs the image from the received blocks. For demonstrative purposes, the original image before watermarking and the image reconstructed at the receiver for = 1 are available in Fig. 10. While we use the Cox algorithm which is not proven to be robust, we reiterate that depending on the data type and application, any robust watermarking scheme can be used in our protocol for that specific application. Works such as [36], [37], [24] present different audio watermarking schemes while works like [35], [46] deal with robust video watermarking. For software watermarking, schemes suggested in [44], [33] can be considered.

Cryptographic Module - DOT For the cryptographic part, we use the RELIC library [8]. The receiver's key is 256-bit long. The sender breaks the document into blocks, encrypts each of the watermarked document and forwards the blocks to the receiver in the first step of DOT protocol. The encryption used to for this step is AES in the counter mode. The sender generates group elements while participating in the DOT protocol to transfer the blocks which are ElGamal encrypted, which are later re-randomized by the receiver. The receiver decrypts the AES encrypted document blocks with the keys obtained through the ElGamal encryption and oblivious transfer.

Table 1 provides the computation timing details for the complete protocol i.e., the time including breaking the document into blocks to the point where the receiver reconstructs the document from received watermarked blocks. It presents the statistics of execution times taken over 100 runs of the experiment. Notice that the timing values reported are when the process is running in a single-thread. With multi-threading and pre-processing ElGamal encryption exponentiation, we expect significant improvement in performance and reduction in timing. Note that each individual step of the transfer protocol can be highly parallelized. While transferring a large volume of data, the sender can divide the data into multiple parts and run the transfer protocol on each part such that copies of the key are watermarked in each part. Each instance of transfer protocol can be run in parallel.

To simulate the dishonest breach and eventual procurement of the leaked document by the receiver, the reconstructed image is sent to the sender of the document. The sender runs the key-extraction algorithm on the obtained image and extracts the receiver's key to perform the penalization.

9 Discussion

9.1 Multiple Receivers

In a scenario involving multiple receivers of the same document, the sender can embed the signing key of a each receiver multiple times into each receiver's version of the document. He can do so by dividing the document into higher number of parts compared to the receiver's key length. This ensures that, in case of collusion and each receiver contributing a small portion of his document while colluding, the sender can still extract considerable amounts of signing keys from the revealed document.

Contracts In Sect. 3, we developed a penalization smart contract for the Bitcoin scripting system, which intentionally has a limited set of instructions. Systems like Ethereum [5] expand this set of instructions into a fully-featured programming language allowing it to perform much elaborate tasks where it is easily possible to write our claim-or-refund contract. However, despite the much better expressivity, it

does not seem to be possible to create an elaborate contact that can *efficiently* substitute the required DOT protocol and robust watermarking scheme. We implemented the penalizing claim-or-refund smart contract as a Bitcoin smart contract as well as a Hyperledger chaincode, as they allow the systems to be executed in a permissionless as well as permissioned blockchain setting. In the future, it would be interesting to create similar solutions using Solidity over the Ethereum network that can at least partially reduce the required cryptographic tools.

Fairness The receiver deposits the bitcoins before the commencement of the protocol and so, if the document transfer does not go through, his funds will be locked till the end of the deposit time period. This is not 'fair' for the receiver. However, in a more realistic setting, in such a scenario the parties would just re-run the protocol and transfer the document.

In the applications where the protocol is used, the sender and the receiver identify themselves to each other with publicly verifiable identities and hence are aware of the identities of the parties they are interacting with. The sender of the data can not abort the protocol multiple times in a wide spread manner as the receiver will simply stop interacting with the sender. He can do that upon noticing the protocol being aborted more than certain number of times there by locking his funds.

Miner The receiver can indeed be a miner in a Bitcoin system. He can try to pre-mine transactions to escape penalty incase of disclosure. This scenario can be prevented by the approach taken in [43, Sec. 6]. In case the sender has the knowledge only of the breach without having access to the revealed document, he can choose to make the watermarking algorithm's private key public to make the receiver lose her deposit.

*Data Custody*In case of storing data at a custodian, the user should be retrieving or downloading the data after the end of time period, this is because if the sender retrieves the data, he can get a copy of the receiver's data with receiver's key embedded in it, he may reveal it to the public and try to blame the receiver for the leak. In such a scenario, the parties can agree to retrieve the deposit and nullify the contract and when the sender decides to store the data again, can perform the protocol. Another way is to have a mechanism in which along with the cooperation of the sender, the receiver can forward a copy of the data with the watermark stripped, such an approach can be looked at in the future.

10 Related work

A closely related subject to penalizing data breaches, one that is well-studied, is traitor tracing [16, 18]. In a traitor tracing scheme, decryption boxes with unique private keys (for a common public key) are distributed to a number of subscribers. If a device is reverse-engineered and the key is leaked, the device it came from can be determined by the service provider.

Kiayais et al. [32] propose a communication optimal asymmetric fingerprinting protocol where the sender forwards different versions of data to the receivers. If any subset of users/receivers collude to produce a pirate copy of the original file/data, the sender can implicate at least one of the colluding parties to a third party judge. The proposed protocol is communication rate (ratio of size of data to the size of transmission length) optimal protocol based on Tardos codes. Each of the receivers is associated with code word and the sent message is divided into parts with two versions of each part. The receivers obtain parts of the data corresponding to the symbols of the codeword. When they collude, the resulting data corresponds to a codeword which is implicated. This approach requires the total document to be reconstructed from the collusion to recover the codeword . Dwork et al. [23] introduce the idea of using secret information (like a credit card number) which is revealed to the other party upon unauthorized sharing of the data. The aim of the work was to prevent the reduction in the amount of data transfer for unauthorized sharing. The adversarial party either has to transfer all the data or lose their secret information to the third party. Kiayias and Tang [30] propose leakage deterring cryptographic primitive schemes where any unauthorized use or transfer of the primitive can result in the secret data (key) being revealed which is embedded in the public data (key) associated with the primitive. However these works require the complete data to be revealed after collusion or use of the primitive. In this work, we consider the two party scenario and focus on the partial leakage of the data which makes it challenging.

Kiayias and Tang [31] add a Bitcoin smart contract to hold a bond that is recoverable. This body of work has limited applicability to our Pepal: problem for three main reasons: (1) we want to detect leaked documents that have been meaningfully written, not keys which are arbitrary, random values; (2) we want the entity distributing the values to not learn the value until it is leaked; and (3) unlike in the smart contract variant [31], we cannot have the provider provision the signing key for use by both parties. For these reasons, we do not build our solution from traitor tracing schemes.

In another line of work, Nasir et al. build a seller-buyer watermarking scheme in [39] where the watermark embedded in the document is not known to seller/sender but can identify the buyer once the document is distributed. The main drawback of their scheme is the requirement of a third trusted authority for providing the watermark for the buyer, also the sender needs to go through the legal procedure and prove to the judge that the buyer is indeed the one who leaked and the penalization is through court system.

In [10], Andre et al. propose a zero-knowledge proof based protocol for providing proof of ownership of the document but does not involve proving that a certain party is the leaker or a way to penalize the leak.

Using bitcoin contracts for collatorizing the fair and correct execution of cryptographic protocols has been explored earlier [12, 15, 34]. Our bitcoin contract is a standard claim-or-refund transaction common in this literature. The main difference is that one party must prove that the signing key used in this transaction is consistent with the one taken as input to a private computation.

11 Conclusion

In this work, we devise and implement the Pepal: protocol that disincentives intentional or unintentional multimedia breach through automated penalization. Our aim here is to raise the bar for the data receivers/custodians by introducing a complementary security mechanism that is inexpensive, automated, and is not restricted by the geo-political boundaries.

To realize our protocol, we have employed robust watermarking and a claim-of-refund smart contract, and proposed a new primitive called Doubly Oblivious Transfer (DOT). DOT along with committed receiver oblivious transfer (CROT) not only ensures that the signing key used by the receiver for the deposit is same as the one used to obtain the document, but also provides no information to the receiver about which of her signing key bits has been embedded in a certain document part. We implement the protocol and observed it to be practical and easy to deploy.

Funding This work has been supported by National Science Foundation (NSF) under grant CNS-1846316.

Availability of data and material Any further data is available on request from the authors.

Code Availability The code for the project can be found at https://github.com/easwarvivek/Pepal

Declarations

Conflicts of interest The authors of this article declare that they have no conflict of interest.

Appendix A: 1-out-of-2 Verified Simplest Oblivious Transfer:

In this protocol, by Doerner et.al. [22] (an augmented version of Oblivious Transfer by Chou et al. [19]), given a multiplicative group G and its generator g, the sender initially chooses a random value $a \leftarrow_R \mathsf{Z}_q$ and the receiver chooses a random value $r \leftarrow_R \mathsf{Z}_q$. The sender transmits $h = g^a$ to the receiver who computes $c = g^{ab+r}$ and transmits to the sender. The sender then computes two keys k_0 and k_1 as $k_0 = H(c^a)$ and $k_1 = H(ch^{-1})^a$ and computes a challenge $p = H(H(k_0)) \oplus H(H(k_1))$ and forwards it to the receiver. The receiver computes the key $k_b = H(h^r)$ and returns $p = H(k_b) \oplus pb$. After verifying if $p = H(H(k_j))$, the sender encrypts M_0 and M_1 using these two keys generating C_0 and C_1 which are then forwarded to the receiver. The receiver decrypts the message M_b using the key $k_c = h^r$. Depending on b, only one of k_0 and k_1 would be equal to g^{ar} computed by the receiver. The other key g^{ar-r^2} can not be computed by the receiver and hence learns no information about M_{b-1}. As the sender just encrypts and forwards the two messages, learns no information about the bit b. Figure 11 provides the depiction of the protocol. The advantage of adding the verification step is that it forces the receiver to compute the keys before receiving the encryptions and makes the protocol (UC)secure in the real-world ideal paradigm.

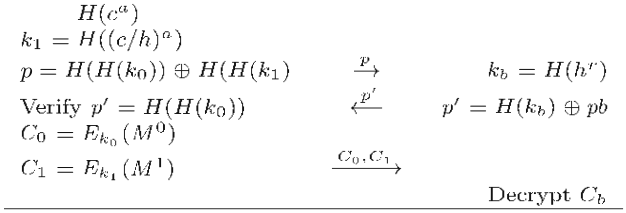


Fig. 11 1-out-of-2 Oblivious Transfer [22]

Functionality $\mathcal{F}_{ZK}^{DL}$ [22] The functionality is parameterized by group G and runs with two parties P_1 and P_2. The parties can be sender S and receiver R.

Proof: On receiving (prove, a, g) where $a \in \mathsf{Z}_q$, g $\in$ G from party P_i, store this message. On receiving, (prove, h, g) from party P_j, where h, g $\in$ G, if $h = g^a$, send (accept) to P_j, otherwise send fail to P_j.

The parties can be sender S and receiver R. The parties use the functionality $\mathcal{F}_{ZK}^{DL}$ to prove in zero-knowledge, that they own the secret keys of the corresponding public keys.

Appendix B: Forwarded Proofs for DOT, CROT and Pepal:

Theorem 1 *The DOT protocol UC-realizes the functionality $\mathcal{F}_{DOT}$ in the $\mathcal{F}_{OT}$-hybrid model under the following conditions:*

Corruption Model: *Static corruption (the sender or receiver is corrupted at the beginning of the protocol).*

Hybrid Functionalities: *H is modelled as a random oracle and secure channels between the parties are assumed.*
Computational Assumption: *The encryption scheme used in the initial step is symmetric, non-committing and robust [19]. Group used for* OT_1^2 *module* G *is a Gap-DH group.*

Proof We prove the security of DOT by constructing a simulator which generates an indistinguishable view in the real world - ideal world paradigm for the adversary. The parties use the functionality F_{ZK}^{DL} to prove in zero-knowledge, that they own the secret keys of the corresponding public keys.

Malicious Sender

- Receive *(*prove*,* sk_S*,* pk_S*)* on behalf of F_{ZK}^{DL}. On accepting, forward accept to the sender, else abort.
- Answer all oracle queries of the sender randomly and store the query and reply pairs in the form of (q_k, r_k).
- Receive the encrypted messages Enc_i, $i \in \{0, 1\}$ from the sender and participate in oblivious transfer for the next step.
- Set the bits $s_i, i \in \{0, 1\}$ randomly with values from $\{0, 1\}$ as choice bits before participating in the OT_1^2 protocol.
- For OT_1^2 part of the protocol, invoke multiple instances corrupted sender phase of the simulator of the UC-secure OT [22] developed by Chou et al. [19, 27] (call it, S_{OT}). The simulator S_{OT} extracts the sender inputs for each of the instances; obtain the inputs.
- Perform the operations like an honest receiver. Receive the elements u_{i,s_i} and try to decrypt (own layer of encryption, the sender is expected to encrypt the messages with $E_{pk}(.)$).
- If any of the received elements results in an error during decryption, abort. Else, re-randomize the encryption using $R_{pk}(.)$ to obtain v_{i,s_i} and forward them back to the sender. Receive an encrypted group element as x_{c,s_c}, try to decrypt and hash it to obtain the decryption key. Decrypt one of the received messages with the obtained key. If it results in an error, abort.
- Decrypt the initial Enc_i as follows: for each i, k, from the initially stored pairs (q_k, r_k), perform $Dec_{r_k}(Enc_i)$. The first value that gets decrypted meaningfully is set as M_i for any i. If no key r_k decrypts meaningfully, set $M_i = \perp$.
- Obtain the choice bit c of the sender as follows: during the OT_1^2 protocol, the simulator S_{OT} extracts the message inputs of the sender side [19] and forwards them to S_{DOT}. For each OT_1^2 instance i, S_{DOT} receives two messages $g_{i,0}, g_{i,1}$ from S_{OT}, the simulator S_{DOT} stores all the elements in the form of $g_{i,j}$. For each i, the simulator checks which of the elements $g_{i,j}$, $j \in \{0, 1\}$, matches with the decrypted element (obtained from sender in the last step of the protocol). Whenever a match is seen, c is set to i.
- Forward the messages $M_i, i \in \{0, 1\}$ and choice bit c to the ideal functionality F_{DOT}.

The adversary can not distinguish between a real world view and simulated view owing to the following facts: the simulator S_{OT} is UC-Secure [22]; ElGamal encryption offers semantic security when DDH is hard; the real world honest receiver's output will be different only if the simulator decrypts the encryptions received to a different value apart from the ones used by the sender, but this happens with a negligible probability owing to the robustness of the encryption scheme.

Malicious Receiver

- Receive *(*prove*,* sk_R*,* pk_R*)* on behalf of F_{ZK}^{DL}. On accepting, forward accept to the receiver, else abort.
- Generate two strings $C_1 \leftarrow A_1(1^\lambda)$ and $C_2 \leftarrow A_1(1^\lambda)$ and forward to the receiver.
- Sample four group elements $g_{i,j}$ for $i, j \in \{0, 1\}$ and encrypt them using ElGamal encryption $E_{pk}(.)$ to obtain $u_{i,j}$.
- Performs two instances of OT_1^2 and use $u_{i,j}$ as inputs for instance i of OT_1^2.
- The receiver inputs s_i to the OT_1^2 instance i. For the OT_1^2 protocol, the simulator invokes the corrupted receiver phase of simulator of Verified Simplest Oblivious Transfer [22] (call it S_{OT}).
- Obtain re-randomized elements v_{i,s_i}, decrypt own layer of encryption using $D_{sk_S}()$ to obtain x_{i,s_i} and forward x_{c,s_c} for a randomly chosen bit c.
- Answer all oracle queries randomly except at the points $g_{i,j}$. When queried on any of the points $g_{i,j}$, sends the bits j, j to the functionality and obtain the message m. • Reply to the query with a key $k \leftarrow A_2(C_p, m)$ where p is uniformly picked from $\{1, 2\}$ for every instance of the simulation.

The receiver can not distinguish the real and simulated view. This is because: ElGamal encryption offers semantic security when DDH is hard, OT_1^2 used is UC-secure [22] and the fact that when the simulator does not abort, the indistinguishability holds from non-committing property of the encryption scheme. The UC-security of the DOT follows from Definition 1.

Theorem 2 *The CROT protocol UC-realizes the ideal functionality* F_{CROT} *in the* F_{ZK}^{DL}*-hybrid model under the following assumptions:*

Corruption Model: *static corruption*

Hybrid Functionalities: *H is modeled as a random oracle and authenticated channels between users are assumed.*
Computational Assumptions: G *is Gap-DH. The symmetric encryption used is non-committing and robust.*

Proof The simulator S_{CROT} interposes between a corrupted party and the CROT functionality F_{CROT}. The verified OT is a "Selective-Failure" Oblivious Transfer, in which the sender can guess the choice bit of the receiver and if the guess is correct, he will be notified it is correct and the receiver is not informed of the same. However, in our CROT protocol, all the messages are transferred simultaneously. For the sender to guess the receiver's choice bits, they need to guess all the bits simultaneously. The probability of the sender guessing all the receiver bits correctly is negligible.

Malicious Sender

The simulator S_{CROT} interposes between a malicious sender and the CROT functionality F_{CROT}, it outputs the sender's messages $M_{i,0}, M_{i,1}$.

- Receiver $(\texttt{prove}, a, A)$ from sender on behalf of $F_{ZK}^{R_{DL}}$. On receiving $(\texttt{accept}, \texttt{A})$ forward it to the sender, else abort.
- Sample random values $s_i, r_i, i \in [0, \kappa - 2]$ and compute the corresponding $c_i = \mathsf{g}^{r_i} \mathsf{h}^{s_i}$ and compute $s_{\kappa-1}, r_{\kappa-1}$ such that $\sum_{i=0}^{\kappa-1} = \mathsf{g}^{r} \mathsf{pk}^{a}$ where $r = \sum_{i=0}^{\kappa-1} 2^i r_i$. Compute ZK-PoKs π_i proving the knowledge of r_i, s_i for each $\mathsf{g}^{r_i} \mathsf{h}^{s^i}$. Forward $\mathsf{g}^{r}, c_i, \pi_i$ to the sender.
- Invoke F_{ZK}^{DL} to prove that the sampled bits correspond to the public key pk
- Compute the pads $k_{i,j} = H(c_i \cdot h^{-j})^a$. Compute the expected challenges as $p_i^{\exp} = H(H(k_{i,0})) \| H(H(k_{i,1}))$
- Upon receiving the sender's challenges p_i, If for any i, $p_i = p_i^{\exp}$, then set the $p_i = H(H(k_{i,0}))$ and add $(\texttt{guess}, s_i)$ to the set G; Otherwise, let Q be the set of all queries made by the sender to the random oracle. If there exists queries Q_j such that such that $H(Q_j) = p_i \| H(H(k_{i,1}))$ then set $s_i = 1$. Otherwise set $s_i = 0$. Add guess, s_i to the set G. Send the set G to F_{CROT}. If cheat-undetected is received, send $k_i = H(H(k^{s_i}))$ to the sender. Otherwise send $k_i = H(H(k^{s}))$ and halt.
- Upon receiving the cipher texts $C_{i,j}$ decrypt them using $k_{i,j}$ and send them to the functionality F_{CROT}.

Malicious Receiver

The simulator interposes between the ideal functionality F_{CROT} and the malicious receiver. It outputs the choice bits s_i of the receiver and the corresponding message chosen M_{i,s_i}. It makes use of the random oracle H and the functionalities $F_{ZK}^{R_{DL}}$

- Sample $a \in \mathsf{Z}_p$ and compute g^a to the receiver on behalf of the functionality F_{ZK}^{DL}.
- Receive g^r, c_i, π_i from the receiver just like an honest sender. Verify the proofs and abort if any of the forwarded proofs fail.
- Compute the keys $k_{i,j}$ like an honest sender.
- Observe the random oracle queries of the receiver. If the receiver ever queries $k_{i,0}$ set $s_i = 0$. If they every query $k_{i,1}$ set $s_i = 1$. Once b_is are set, send s_i to the the functionality F_{CROT} and receiver no-cheat.
- Run the verification as the honest sender would.
- Upon receiving the messages M_{i,s_i} from the functionality, set the corresponding ciphertexts as $C_{i,s_i} = E_{k_{i,s_i}}(M_{i,s_i})$ and set the other ciphertexts to random values.

In the malicious sender case, the first message received by the consists of the $\mathsf{g}^r, c_i, \mathsf{PoK}$. Since r is picked randomly, the view of the sender is identical in both the worlds. The simulator S_{CROT} receives the value a on behalf of the functionality F_{ZK}^{DL} and so can compute the values r_i, c_i such that the zero-knowledge proof and verification check hold. It can also compute the the keys $k_{i,j}$ and hence verify if the challenges received p_i are correct.

During the verification phase of the transfer, the sender is required to compute values $H(H(k_{i,0}))$, $H(H(k_{i,1}))$, only one set of the hashes are known to the receiver which correspond to $H(H(k_{i,s_i}))$. To induce a selective failure, the sender can try to guess the receiver bits and set random values for the opposite ones while calculating the challenges p_i, To guess all the bits correctly and simultaneously, the sender succeeds only with negligible probability $\frac{1}{2^\kappa}$. All the oracle queries made by the sender can be used to compute the sender's guesses in the protocol which can be forwarded to the functionality which aborts if the guesses are incorrect. After this point, the simulator behaves like an honest real world receiver and forwards all the messages accordingly and aborts under same conditions. There the view of the malicious sender under the real world execution of the protocol is indistinguishable from its view while interacting with the simulator S_{CROT}, he can distinguish the view with no better probability than $\frac{2l}{\kappa}$.

In the malicious receiver case, h is chosen by the simulator and c_i is chosen by the receiver. These values fix the computed keys $k_{i,j}$ to be computed. The receiver can not guess the k_{i,s_i} values except with probability of $\frac{1}{2^\kappa}$ for each. When the receiver queries the random oracle, the simulator records the queries and finds the corresponding choice bit s_i. If the receiver can query the random oracle at k_{i,s_i} and $k_{i,1-s_i}$, then the simulator can not compute the choice bit. However the receiver can not make both those queries, as any such receiver breaks the CDH assumption. The rest of the simu-

lator steps follow a honest sender and the view generated is identically distributed to the real-world paradigm. Thus the view of the malicious receiver is identical in the real world and the ideal world paradigm if the CDH problem is hard in the group selected.

Theorem 3 *The Pepal: protocol securely implements the ideal functionality $F_{\text{Pepal:}}$ in the F_{DOT}, F_{ZK}^{DL} hybrid model under the following assumptions:*

Corruption Model: *static corruption*
Hybrid Functionalities: *H is modeled as a random oracle and authenticated channels between users are assumed.*
Computational Assumptions: *CDH and DDH are assumed to be hard in* G, G *is Gap-DH. The symmetric encryption used is non-committing and robust.*

Proof Pepal: protocol uses DOT which internally uses CROT instead of multiple instances of the standard OT_1^2 for the transfer of messages/document blocks from the sender to the receiver. The simulator for the Pepal: protocol simply invokes the corresponding simulator S_{DOT} which invokes the simulator S_{CROT} instead of instances of S_{OT}. The UC-security of the CROT protocol is already established through Theorem 2.

Malicious sender

The simulator S_{CROT} interposes between a malicious sender and the Pepal: functionality $F_{\text{Pepal:}}$.

- Receive (prove, sk_{S}, pk_{S}) on behalf of F_{ZK}^{DL}. On accepting, forward accept to the sender, else abort.
- Sample a random secret key sk_R and parse the bits of the secret key into s_i, $i \in [0, \kappa - 1]$ and participate in the DOT protocol.
- Invoke the malicious sender phase of the simulator S_{DOT} for the same.
- The simulator S_{DOT} receives the ElGamal encryptions from the sender just as a receiver would
- For the message transfer, S_{DOT} inturn invokes a single instance of the malicious sender phase of the CROT simulator S_{CROT} (instead of multiple instances of S_{OT}) during the transfer phase.
- The simulator S_{CROT} after interacting with the malicious sender, outputs the sender messages $u_{i,j}$. Since the simulator acts as the receiver it has access to sk_{R}. It also has access to sk_{S} through the F_{ZK}^{DL} functionality. Hence it can decrypt the messages $u_{i,j}$.
- After this the simulator behaves like a honest receiver and participates in all the further protocol steps.
- The keys $u_{i,j}$ are used to decrypt the messages $M_{i,j}$. Forward the messages $M_{i,j}$ to the functionality $F_{\text{Pepal:}}$ as inputS, $M_{i,j}, \pi, sid$ for the session id sid.

Malicious receiver The simulator S_{CROT} interposes between a malicious receiver and the Pepal: functionality $F_{\text{Pepal:}}$.

- Receive (prove, sk_{R}, pk_{R}) on behalf of F_{ZK}^{DL}. On accepting, forward accept to the receiver, else abort.
- Invoke the malicious receiver phase of the simulator S_{DOT} which forwards the encryptions of the keys.
- As a part of steps of S_{DOT}, invoke the malicious receiver phase of S_{CROT} instead of multiple instances of the simulator S_{OT}.
- S_{CROT} outputs the choice bits s_i of the receiver.
- Forward the choice bits to the functionality F_{DOT} to obtain the messages M_{i,s_i}.
- Use the receiver bits s_i through DOT simulator to set the encryptions which can be opened by the receiver to the values forwarded by the functionality F_{DOT}.

The simulator $S_{\text{Pepal:}}$ is the simulator S_{DOT} which invokes the simulator S_{CROT} instead of multiple instances of S_{OT} for the transfer protocol. The UC-security follows from the UC-security of the DOT and the CROT protocols. S_{DOT} which internally invokes S_{CROT} (instead of S_{OT}), produces an indistinguishable view for the adversary in the real world-ideal world paradigm.

References

1. (2015) Data protection and breach. https://otalliance.org/system/files/files/resource/documents/dpd_2015_guide.pdf
2. (2015) Man in the cloud (mitc) attacks. https://www.imperva.com/docs/HII_Man_In_The_Cloud_Attacks.pdf
3. (n.d.) Data breaches. https://www.privacyrights.org/data-breaches?title=&breach_type%5B%5D=267
4. (n.d.) Digital watermarking alliance. http://digitalwatermarkingalliance.org/
5. (n.d.) Ethereum website. https://www.ethereum.org/
6. (n.d.) Friendmts. https://www.friendmts.com/nab-2017-showcase/
7. (n.d.) Nsw data and information custodianship policy. https://www.finance.nsw.gov.au/ict/sites/default/files/NSW%20Data%20and%20Information%20Custodianship%20Polic%20v1-0.pdf
8. (n.d.) Relic: efficient library for cryptography. https://github.com/relic-toolkit
9. Adelsbach, A., Sadeghi, A.R.: Zero-knowledge watermark detection and proof of ownership. In: Information Hiding (2001a)
10. Adelsbach, A., Sadeghi, A.R.: Zero-knowledge watermark detection and proof of ownership. In: Moskowitz, I.S. (ed.) Information Hiding, pp. 273–288. Springer, Berlin Heidelberg, Berlin, Heidelberg (2001)
11. Amer, I., Sheha, T., Badawy, W., Jullien, G.: A tool for robustness evaluation of image watermarking algorithms. In: Elleithy, K. (ed.) Advanced Techniques in Computing Sciences and Software Engineering, pp. 59–63. Springer, Netherlands, Dordrecht (2010)
12. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: IEEE Symposium on Security and Privacy (2014)

13. Arun, V., Kate, A., Garg, D., Druschel, P., Bhattacharjee, B.: Finding safety in numbers with secure allegation escrows arXiv preprint arXiv:1810.10123 (2020)
14. Bast, C.M.: At what price silence: are confidentiality agreements enforceable? William Mitchell Law Rev. **25**(2), 627 (1999)
15. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: ICC (2014)
16. Boneh, D., Franklin, M.: An efficient public key traitor tracing scheme. In: CRYPTO (1999)
17. Camenisch, J., Stadler, M.: Proof Systems for General Statements About Discrete Logarithms, p. 260. Technical report/Dept of Computer Science, ETH Zürich (1997)
18. Chor, B., Fiat, A., Naor, M.: Tracing traitors. In: CRYPTO (1994)
19. Chou, T., Orlandi, C.: The simplest protocol for oblivious transfer. In: LATINCRYPT (2015)
20. Cox, I.J., Kilian, J., Leighton, F.T., Shamoon, T.: Secure spread spectrum watermarking for multimedia. IEEE TIP **6**(12), 1673–1687 (1997)
21. Cunningham, T.J., Huffman, B., Salmon, C.M.: Settlement trends in data breach litigation (2014). https://www.financierworldwide.com/settlement-trends-in-data-breach-litigation
22. Doerner, J., Kondi, Y., Lee, E., a shelat.: Secure two-party threshold ecdsa from ecdsa assumptions. In: 2018 IEEE Symposium on Security and Privacy (SP), pp 595–612, (2018) https://doi.org/10.1109/SP.2018.00036
23. Dwork, C., Lotspiech, J., Naor, M.: Digital signets: Self-enforcing protection of digital information (preliminary version). In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp 489–498 (1996)
24. Erfani, Y., Siahpoush, S.: Robust audio watermarking using improved TS echo hiding. Digital Signal Process. **19**(5), 809–814 (2009). https://doi.org/10.1016/j.dsp.2009.04.003
25. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Proceedings on Advances in cryptology—CRYPTO '86, pp 186–194 (1987)
26. Floyd, T., Grieco, M., Reid, E.F.: Mining hospital data breach records: Cyber threats to u.s. hospitals. In: 2016 IEEE Conference on Intelligence and Security Informatics (ISI), pp 43–48 (2016)
27. Genc, Z.A., Iovino, V., Rial, A.: The simplest protocol for oblivious transfer revisited (2017). https://eprint.iacr.org/2017/370.pdf
28. Härder, T., Bühmann, A.: Database caching-towards a cost model for populating cache groups. In: Benczúr, A., Demetrovics, J., Gottlob, G. (eds.) Advances in Databases and Information Systems, pp. 215–229. Springer, Heidelberg (2004)
29. Hourihan, C., Cline, B.: A look back: U.s. healthcare data breach trends". (2008) https://hitrustalliance.net/content/uploads/2014/05/HITRUST-Report-U.S.-Healthcare-Data-Breach-Trends.pdf
30. Kiayias, A., Tang, Q.: How to keep a secret: leakage deter-ring public-key cryptosystems. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, pp 943–954 (2013)
31. Kiayias, A., Tang, Q.: Traitor deterring schemes: using bitcoin as collateral for digital content. In: ACM CCS (2015)
32. Kiayias, A., Leonardos, N., Lipmaa, H., Pavlyk, K., Tang, Q.: Communication optimal tardos-based asymmetric fingerprinting. In: Nyberg, K. (ed.) Topics in Cryptology – CT-RSA 2015, pp. 469–486. Springer International Publishing, Cham (2015)
33. Kim, S., Wu, D.J.: Watermarking cryptographic functionalities from standard lattice assumptions. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017, pp. 503–536. Springer International Publishing, Cham (2017)
34. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: IEEE Symposium on Security and Privacy (2016)
35. Lancini, R., Mapelli, F., Tubaro, S.: A robust video watermarking technique in the spatial domain. In: International Symposium on VIPromCom Video/Image Processing and Multimedia Communications, pp 251–256, (2002) https://doi.org/10.1109/VIPROM.2002.1026664
36. Lei, B.Y., Soon, I.Y., Li, Z.: Blind and robust audio watermarking scheme based on svd-dct. Signal Process. **91**(8), 1973–1984 (2011). https://doi.org/10.1016/j.sigpro.2011.03.001
37. Lie, W.N., Chang, L.C.: Robust and high-quality time-domain audio watermarking based on low-frequency amplitude modification. IEEE Trans. Multimed. **8**(1), 46–59 (2006). https://doi.org/10.1109/TMM.2005.861292
38. Meerwald, P.: Watermarking source code. Online, (2005) http://www.cosy.sbg.ac.at/~pmeerw/Watermarking
39. Memon, N., Wong, P.W.: A buyer-seller watermarking protocol. IEEE Trans. Image Process. **10**(4), 643–649 (2001). https://doi.org/10.1109/83.913598
40. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
41. Rahulamathavan, Y., Rajarajan, M., Rana, O.F., Awan, M.S., Burnap, P., Das, S.K.: Assessing data breach risk in cloud systems. In: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), pp 363–370 (2015)
42. Rogaway, P.: Formalizing human ignorance. In: Nguyen, P.Q. (ed.) Progress in Cryptology - VIETCRYPT 2006, pp. 211–228. Springer, Heidelberg (2006)
43. Ruffing, T., Kate, A., Schröder, D.: Liar, liar, coins on fire!: penalizing equivocation by loss of bitcoins. In: ACM CCS (2015)
44. Venkatesan, R., Vazirani, V., Sinha, S.: A graph theoretic approach to software watermarking. In: Moskowitz, I.S. (ed.) Information Hiding, pp. 157–168. Springer, Berlin Heidelberg, Berlin, Heidelberg (2001)
45. Yao, S.B.: An attribute based model for database access cost analysis. ACM Trans. Database Syst. **2**(1), 45–67 (1977). https://doi.org/10.1145/320521.320535
46. Zhang, J., Ho, A.T.S., Qiu, G., Marziliano, P.: Robust video watermarking of h.264/avc. In: IEEE Transactions on Circuits and Systems II: Express Briefs 54(2):205–209, (2007) https://doi.org/10.1109/TCSII.2006.886247