2019 IEEE European Symposium on Security and Privacy Workshops

Towards Automatically Penalizing Multimedia Breaches

(Extended Abstract)

Easwar Vivek Mangipudi
Purdue University
emangipu@purdue.edu

Krutarth Rao
Hewlett Packard
raokrutarth@gmail.com

Jeremy Clark
Concordia University
j.clark@concordia.ca

Aniket Kate
Purdue University
aniket@purdue.edu

***Abstract*—This work[1] studies the problem of automatically penalizing intentional or unintentional data breach (APB) by a receiver/custodian receiving confidential data from a sender. We solve this problem for multimedia data by augmenting a blockchain *on-chain* smart contract between the sender and receiver with an *off-chain* cryptographic protocol, such that any significant data breach from the receiver is penalized through a monetary loss. Towards achieving the goal, we use an oblivious transfer scheme when used with a robust watermarking and a claim-or-refund blockchain contract provides the necessary framework to realize the APB protocol in a provably secure manner. In our APB protocol, a public data breach by the receiver leads to her Bitcoin (or other blockchain) private signing key getting revealed to the sender, which allows him to penalize the receiver by claiming the deposit from the claim-or-refund contract. Interestingly, the protocol also ensures that the malicious sender cannot steal the deposit, even as he knows the original multimedia document or releases it in any form.**

I. Introduction

Data breach attacks on cloud hosts have become common place [1], [2], [3], [4], [5], 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 these data breaches can be avoided with good security practices on the custodian's infrastructure [6], there is no evident decrease in the number. In these cases, taking legal actions is not only expensive and time consuming but it is also difficult to establish the responsibility in today's geo-politically distributed data flows.

This work aims at *raising 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. In particular, our goal is to make the data custodians more accountable through automatically enforceable monetary penalties resulting in immediate loss of funds, and we call the associated contract the *automated penalization of breach* (APB) contract. Applicability scenarios for APB contracts range from industrial media custodianship, 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 [7].

[1]The full version of this work can be found at https://eprint.iacr.org/2018/1050.pdf

Example Scenario I – Data Custodianship: Data Custodianship refers to the responsibility of safe storage and custody of the data [8]. A serious breach of the storage typically results in criminal litigation against the custodian. APB can be useful when legal action is undesirable due to the uncertainty of recovering the payment (which increases if the winning party is owed court costs in addition to the actual remedy) [9]. We assume that the data owner/sender and the custodian/receiver agree on an amount of money that will be awarded to the owner should specified documents be demonstrably leaked by the custodian. Towards automatically ensuring that the owner will receive the funds, this amount could take the form of a surety bond that is held in trust by a Bitcoin or other permission-less/permissioned blockchain based cryptocurrency smart contract.

Example Scenario II – Media Download: In the case of a users downloading paid media that should not be publicly shared on online platforms, the downloaders make a timed deposit along with the actual payment, for a agreed upon time and value for the download. The deposit can be forfeited by the media provider upon dishonest sharing of the content, in case of no such incident it will be returned to the downloader/customer.

APB 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 important as some disclosures are in the public interest (whistle-blowing) [10]. In fact, in certain cases, 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 APB protocol is a step in that direction.

Contributions. In the form of APB, we formalize the problem of automatically settling intentional or unintentional data breaches with a Bitcoin (or other blockchain) smart contract, eschewing the traditional recourse of costly legal action. Our APB protocol is a crypto-augmented smart contract system to obtain an arbitrator-free settlement. It consists of four main components: a claim-or-refund smart contract, a robust watermarking scheme, a oblivious-transfer scheme and

DOI 10.1109/EuroSPW.2019.00044

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 [11], [12], [13] 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 APB 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 protocol securely realizes 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.

II. An Overview of APB

Problem Definition. We consider a setting where a sender wishes to disclose a 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 a 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. The receiver's Bitcoin wallet holds the 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 so as to steal the deposit. When appropriate, he can also make the document public and try to accuse the receiver of dishonest disclosure. The malicious receiver, on the other hand, 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:

- *Sender Privacy*: Before the transfer completes, no information regarding the document is available to the receiver.
- *Receiver Privacy*: Before the disclosure of 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 Figure 4 in Section V.

Solution Overview. We propose the APB protocol, depicted in Figure 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 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_1^2s 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 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 signing-key of the receiver. However, if the document is revealed/disclosed before the time of expiry of the agreement, the sender learns the signing key of the receiver from the watermark of the revealed document. He can then proceed to 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 would expire after the agreed time and the funds will be transferred back to her. The transactional logic of the deposit is depicted as pseudo-code in Algorithm 1.

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

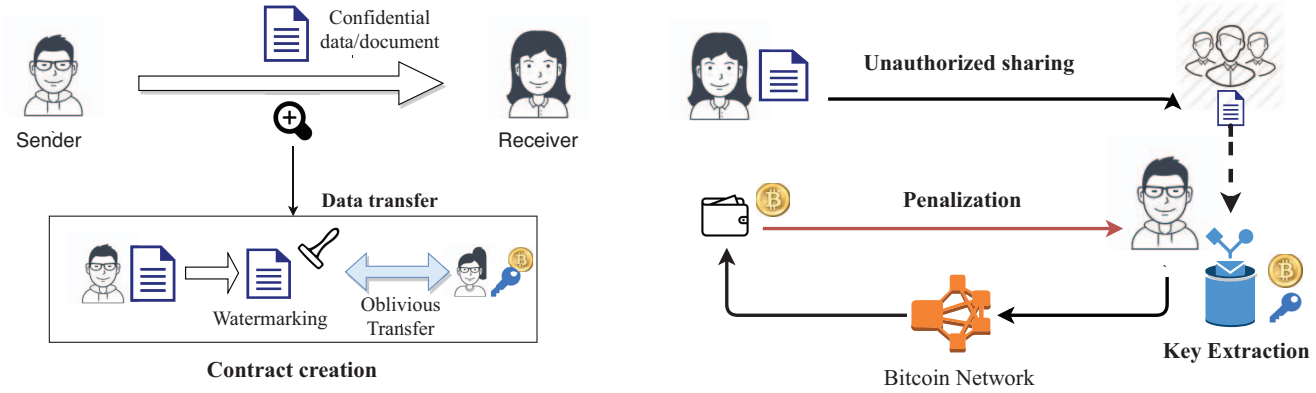

Fig. 1: APB Protocol: High-level View

Algorithm 1 Claim-or-Refund contract logic

```
1: if Current time t_now ≥ t then
2:     Direct the locked funds back to the contract creator
3: else
4:     if Both the sender and receiver sign the transaction then
5:         Direct the funds to the mentioned recipient
6:     else
7:         Transaction is invalid
```

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 number of parts at those document locations.

Notice that our APB 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 [14]. However, defining the complete solution as a smart contract will not be or may not remain inexpensive enough.

III. Functional Blocks

Robust Bit Watermarking. Once the dishonest receiver reveals the document, the sender learns the signing-key using watermark of the revealed document. For watermarking document blocks, we use a *robust* bit watermarking scheme with the property that the watermarked bit 0 or 1 cannot be removed without loss of significant information from the block. The actual watermarking scheme used can vary based on the type of the document being watermarked. We mostly follow the definition of robust watermarking by Adelsbach et al. [15].

Let $\mathcal{M}$ denote the set of all documents, $\mathcal{WM} \subseteq \{0,1\}$ the set of two watermarks. $\mathcal{K}$ indicates the set of all keys and λ indicates the security parameter. The watermarking scheme is defined using three algorithms, one each for key generation, embedding and detection of the watermark. *Gen* (λ) is a probabilistic algorithm that outputs a key $k \in \mathcal{K}$ for the given λ. *Embed* (M, w, k) takes the multimedia document M, watermark $w \in \mathcal{WM}$ and key k as inputs and generates a watermarked document M' while *Detect* (M', k, w) takes the watermarked document M', the key k and the watermark w as input and outputs $\top$ if the watermark in M' matches w, else outputs $\perp$. For non-blind watermarking schemes, the detection algorithm takes the original document M also as input.

We require the watermarking scheme to satisfy the three properties of *Effectiveness, Robustness* and a weaker version of imperceptibility (as in [15]) called *Bit-imperceptibility*. *Effectiveness* indicates that a key k used to embed a watermark should also detect the watermark i.e., $\forall M \in \mathcal{M}, \forall k \in \mathcal{K}$ and $\forall w \in \mathcal{WM}$, if $Embed(M, w, k) \rightarrow M'$, then $Detect(M', k, w) = \top$. *Robustness* states that no probabilistic polynomial-time (PPT) adversary should be able to effectively change or remove the watermark in the watermarked document without leaving the document itself unusable. *Bit-imperceptibility* indicates that the knowledge of the watermarked document with some unknown watermark bit $w \in \mathcal{WM}$ should not reveal any additional information on the watermark bit that can be feasibly extracted.

Oblivious Transfer. 1-out-of-2 oblivious transfer (OT_1^2) 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.* [16] which is an extended version of OT protocol by Chou et.al. [17]. The multiplicative group G used for the protocol is Gap-DH [18] 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 [19] 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
pk_R OP_CHECKSIGVERIFY
ELSE
OP_2 pk_R pk_S OP_2 OP_CHECKMULTISIGVERIFY
ENDIF

IV. Committed Receiver Oblivious Transfer

Oblivious Transfer is used to transfer one message M_b where $b \in \{0,1\}$ of the two messages M_0 and M_1 from sender to the receiver with bit b. However in our APB protocol, 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 Figure 2. We depict the construction of the protocol in Figure 3.

Theorem 1: The *CROT* protocol securely implements the ideal functionality $\mathcal{F}_{CROT}$ under the following assumptions:

$\mathcal{F}_{CROT}$ **interacts with sender** S **and receiver** R**.**
- R generates the key pair (sk, pk) and forwards the bits $s_i \in \{0,1\}, i \in [0, \cdots, \kappa-1]$ and the κ bit signing-key sk to $\mathcal{F}_{CROT}$ which stores them.
- S forwards the messages $M_{i,0}, M_{i,1}; i \in [0, \cdots, \kappa-1]$ to $\mathcal{F}_{CROT}$ which stores them.
- After receiving the inputs from both S and R, $\mathcal{F}_{CROT}$ verifies if the bits of sk are the bits s_i and (pk, sk) are a key pair. If the verification succeeds, it forwards the messages M_{i,s_i}, $i \in [0, \cdots, \kappa-1]$ to the receiver, else, aborts.

Fig. 2: Ideal Functionality $\mathcal{F}_{CROT}$ of *CROT*

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.

Construction

The protocol construction for the ideal functionality $\mathcal{F}_{CROT}$ as given is the Figure 2 is presented here. The sender has messages $M_{i,j}$ for $0 \leq i \leq \kappa-1$ and $j \in \{0,1\}$. The receiver has a signing key sk (s_i for $0 \leq i \leq \kappa-1$ are the bits of sk). Given a multiplicative group G and its generator g, the sender initially chooses a random value $a \leftarrow \mathbb{Z}_q$ 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 $0 \leq i \leq \kappa-1$. 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 $0 \leq i \leq \kappa-1$, the receiver forwards non-interactive zero knowledge (NIZK) proofs of knowledge of exponents 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 [20] of an interactive sigma protocol for Pedersen commitments in the random oracle model. Following the formal symbolic notation introduced by Camenisch and Stadler [21], each proof is depicted as $\mathbf{PoK}\{(r_i, s_i)|\mathsf{g}^{r_i}\mathsf{h}^{s_i}\}$ in Figure 3. 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.

After successful verification the sender computes the keys $k_{i,j} = H((\mathsf{c}_i \cdot h^{-j})^a)$ for each $0 \leq i \leq \kappa-1$ and $j \in \{0,1\}$. The sender verifies if the receiver computed the keys using the verification step similar to Verified Simplest OT [16]. 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 [17]) and so can not decrypt $C_{i,1-s_i}$. 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 [18] while the encryption used is symmetric, non-committing and robust [17].

V. The APB protocol

Here, we detail the steps of the APB protocol which uses *CROT*. The watermarking and the OT_1^2 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 section III) can be used.

4) ***CROT***$(M_{i,0}, M_{i,1}, sk)$: The sender watermarks the document blocks to obtain $M_{i,j}$ and inputs them to the *CROT* protocol. The receiver after proving in zero knowledge that the input to the protocol is her signing key sk, receives the encrypted blocks. 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 APB protocol begins, after the two parties agree on the APB 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 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.

Analysis. Figure 4 presents the ideal functionality $\mathcal{F}_{APB}$ for APB, while Theorem 2 proves its security. Here we show that the functionality achieves the desirable properties discussed in Section II. 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 APB, 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 APB protocol.

Theorem 2: The APB protocol securely implements the ideal functionality $\mathcal{F}_{APB}$ 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 $\mathbb{G}$, G is Gap-DH. The symmetric encryption used is non-committing and robust.

Proof Outline of CROT. The security of the protocol directly follows from the fact that the OT_1^2 protocol [16] is UC-secure under Gap-DH and ZK proof of knowledge of exponent forwarded by the receiver does not leak any information regarding s_i to the sender. Hence *CROT* is UC-secure. Since APB uses only *CROT* for the transfer, APB is UC-secure.

A. Discussion

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

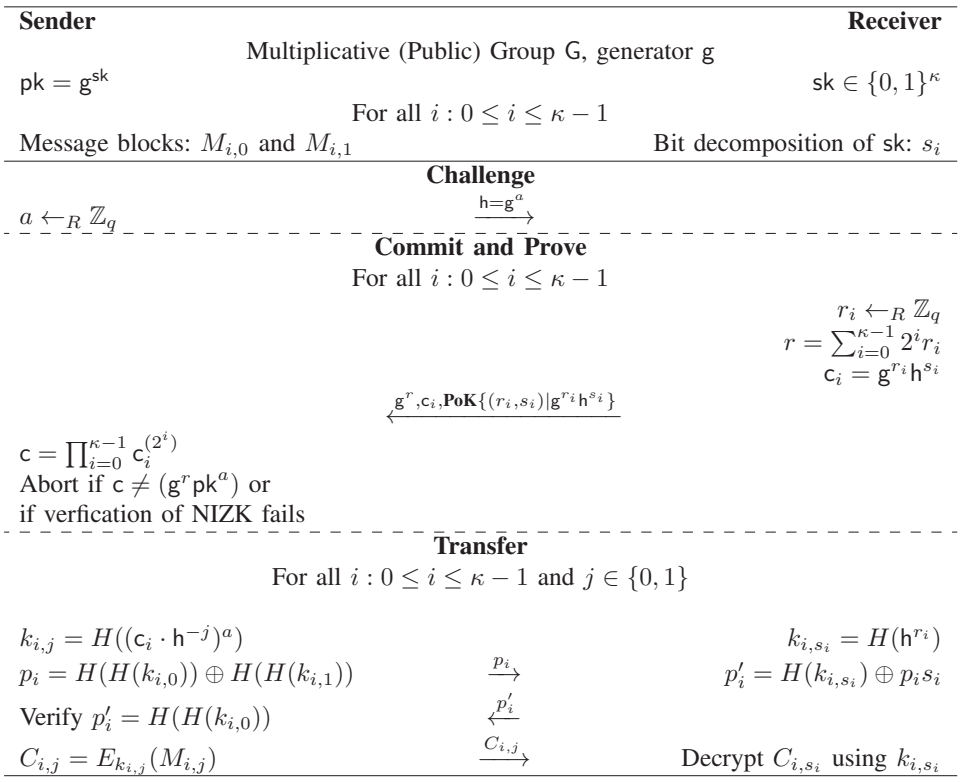

Fig. 3: Committed Receiver Oblivious Transfer (CROT) Protocol

$\mathcal{F}_{APB}$ **interacts with sender** S **and receiver** R**.**
- The receiver R generates the key pair (sk, pk), forwards the bits $s_i \in \{0,1\}, i \in [0, \cdots, \kappa - 1]$ of the κ bit signing-key sk and sk to the functionality which stores the received input.
- The sender S forwards the *watermarked* document blocks $M_{i,0}, M_{i,1} \in \{0,1\}^*, i \in [0, \cdots, \gamma - 1]$ to the functionality which stores the received input.
- The functionality verifies if the bits of sk are the bits s_i, if the verification succeeds, it forwards the message blocks M_{i,s_i} to R, else, it aborts.

Fig. 4: Ideal Functionality $\mathcal{F}_{APB}$ of APB

while colluding, the sender can still extract considerable amounts of signing keys from the revealed document.

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.

VI. Related work and Further Discussion

We are unaware of any academic research into cryptographically-enforceable automated penalization of a data breach. A closely related subject, one that is well-studied, is traitor tracing [22], [23]. 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. A recent proposal by Kiayias and Tang [24] adds a Bitcoin smart contract to hold a bond that is recoverable. This body of work has limited applicability to our APB 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 [24], 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 [25] 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 [26] 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 [11], [12], [27]. 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 singing key used in this transaction is consistent with the one taken as input to a private computation.

VII. Conclusion

In this work, we devise the APB 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, a claim-of-refund smart contract and a oblivious transfer protocol. Our protocol ensures that the sender can extract the signing key of the receiver from the disclosed document while ensuring that the key used by the receiver for the deposit is same as the one used for obtaining the document.

References

[1] "Statista. annual number of data breaches and exposed records in the united states from 2005 to 2018," www.statista.com/statistics/273550/data-breaches-recorded-in-the-united-states-by-number-of-breaches-and-records-exposed/.

[2] "Healthcare it news. 128, 000 records breached in ransomware attack on arkansas oral facial surgery centre (2017)," https://www.healthcareitnews.com/news/ransomware-attack-breaches-128000-patient-records-arkansas-provider.

[3] "Healthcare informatics. 4.4m patient records breached in q3 2018, protenus finds," https://www.hcinnovationgroup.com/cybersecurity/news/13030867/44m-patient-records-breached-in-q3-2018-protenus-finds.

[4] "Man in the cloud (mitc) attacks," https://www.imperva.com/docs/HII_Man_In_The_Cloud_Attacks.pdf, 2015.

[5] T. Floyd, M. Grieco, and E. F. Reid, "Mining hospital data breach records: Cyber threats to u.s. hospitals," in *2016 IEEE Conference on Intelligence and Security Informatics (ISI)*, Sept 2016, pp. 43–48.

[6] "Data protection and breach," https://otalliance.org/system/files/files/resource/documents/dpd_2015_guide.pdf, 2015.

[7] "Data breaches," https://www.privacyrights.org/data-breaches?title=&breach_type%5B%5D=267.

[8] "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.

[9] T. J. Cunningham, B. Huffman, and C. M. Salmon, "Settlement trends in data breach litigation," https://www.financierworldwide.com/settlement-trends-in-data-breach-litigation, 2014.

[10] C. M. Bast, "At what price silence: are confidentiality agreements enforceable?" *William Mitchell Law Review*, vol. 25, no. 2, 1999.

[11] I. Bentov and R. Kumaresan, "How to use bitcoin to design fair protocols," in *ICC*, 2014.

[12] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek, "Secure multiparty computations on bitcoin," in *IEEE Symposium on Security and Privacy*, 2014.

[13] T. Ruffing, A. Kate, and D. Schröder, "Liar, liar, coins on fire!: Penalizing equivocation by loss of bitcoins," in *ACM CCS*, 2015.

[14] "Ethereum Website," https://www.ethereum.org/.

[15] A. Adelsbach, S. Katzenbeisser, and A. Sadeghi, "A computational model for watermark robustness," in *8th International Workshop Information Hiding (IH)*, 2006, pp. 145–160.

[16] J. Doerner, Y. Kondi, E. Lee, and a. shelat, "Secure two-party threshold ecdsa from ecdsa assumptions," in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 595–612. [Online]. Available: doi.ieeecomputersociety.org/10.1109/SP.2018.00036

[17] T. Chou and C. Orlandi, "The simplest protocol for oblivious transfer," in *LATINCRYPT*, 2015.

[18] Z. A. Genc, V. Iovino, and A. Rial, "The simplest protocol for oblivious transfer revisited." [Online]. Available: https://eprint.iacr.org/2017/370.pdf

[19] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[20] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Proceedings on Advances in cryptology—CRYPTO '86*, 1987, pp. 186–194.

[21] J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," *Technical report/Dept. of Computer Science, ETH Zürich*, vol. 260, 1997.

[22] B. Chor, A. Fiat, and M. Naor, "Tracing traitors," in *CRYPTO*, 1994.

[23] D. Boneh and M. Franklin, "An efficient public key traitor tracing scheme," in *CRYPTO*, 1999.

[24] A. Kiayias and Q. Tang, "Traitor deterring schemes: Using bitcoin as collateral for digital content," in *ACM CCS*, 2015.

[25] N. Memon and P. W. Wong, "A buyer-seller watermarking protocol," *IEEE Transactions on Image Processing*, vol. 10, no. 4, pp. 643–649, April 2001.

[26] A. Adelsbach and A.-R. Sadeghi, "Zero-knowledge watermark detection and proof of ownership," in *Information Hiding*, I. S. Moskowitz, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 273–288.

[27] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE Symposium on Security and Privacy*, 2016.