Cryptographic Collateralized Loan without Smart Contracts

Diego Castejon-Molina
IMDEA Software Institute
Universidad Politécnica de Madrid
diego.castejon@imdea.org

Varun Madathil
Yale University
varun.madathil@yale.edu

Dimitrios Vasilopoulos
SnT
University of Luxembourg
dimitrios.vasilopoulos@uni.lu

Sri AravindaKrishnan Thyagarajan
University of Sydney
aravind.thyagarajan@sydney.edu.au

Pedro Moreno-Sanchez
IMDEA Software Institute
MPI-SP
pedro.moreno@imdea.org

Abstract

Cryptocurrency lending is growing rapidly, and smart-contract-based loans are expected to grow further. However, existing systems are fundamentally limited: they only operate on smart-contract-enabled blockchains, and assets from other blockchains can be used only via tokenized representations.

In this work, we propose an oracle-aided cryptographic protocol that implements the logic of collateralized loans without smart contracts, instead only requiring basic transactions from the underlying blockchain and hence, being compatible with limited-scripting blockchains, including Bitcoin. For that, we introduce *verifiable graph encryption for signatures* (VGES), a new cryptographic primitive that, on input a graph modeling the correspondence between transactions for collateral distribution (vertices) and loan repayments (edges), permits to encrypt the signatures on collateral-distribution transactions ensuring: (1) a signature can only be decrypted after completing loan repayments corresponding to a valid path in the graph from the root (*graph enforcement*); and (2) anyone can verify that encrypted signatures are valid and can be decrypted after doing the required loan repayments according to the graph (*verifiability*). We present two provably secure constructions of VGES and the evaluation of our implementation shows that they offer a tradeoff between the number of required on-chain transactions and off-chain computation, while both remain efficient on commodity hardware.

1 Introduction

Blockchain-based cryptocurrencies allow mutually distrustful parties to execute financial applications without reliance on a trusted third party. To this end, cryptocurrencies like Ethereum natively support complex, stateful smart contracts written in expressive high-level languages, enabling a wide range of decentralized finance (DeFi) applications. One prominent example of such applications is *collateralized loans*. A collateralized loan involves two parties, a bank and a client. The bank extends credit to the client (i.e., the *loan*), only if the client provides a security deposit (i.e., the *collateral*), that the bank can seize if the client does not repay the loan. Collateralized loans are highly relevant in practice: as of early 2025, cryptocurrency-based lending accounts for approximately $50B and is expected to continue growing [26]. In addition, financial institutions, such as the European Central Bank, are considering accepting cryptocurrencies as collateral on the securities market [16]. Several DeFi platforms, including Aave [12, 13], Compound [18, 19], Morpho [34], and Liquity [29, 30], leverage Ethereum smart contracts to automate loans.

```
contract SimpleLoan {
    address payable client, bank;
    uint collateralAmt, loanAmt, repaidAmt, dueDate,
         collateralPaid;
    IERC20 token;
    ...
    function borrowLoan()  payable public {
        require(msg.sender == client);
        require(msg.value >= collateralAmt);
        collateralPaid = msg.value;
        token.transfer(client, loanAmt); }

    function repayInstallment(uint amt) public {
        require(msg.sender == client);
        require (amt >= 0);
        token.approve(address(this), amt);
        token.transferFrom(msg.sender, bank, amt);
        repaidAmt += amt; }

    function withdrawCollateral() public {
        if (msg.sender == client) {
            require(block.timestamp <= dueDate);
            client.transfer(repaidAmt);
            bank.transfer(collateralAmt - repaidAmt); }
        else {
            require(msg.sender == bank);
            require(block.timestamp > dueDate);
            bank.transfer(collateralAmt); } } }
```

Figure 1: Example of collateralized loan smart contract.

Smart contract-based collateralized loan. An illustrative example of a smart contract (SC) implementing a collateralized loan is shown in Figure 1. First, the client invokes `borrowLoan` to (i) deposit `collateralAmt` units of Ether into the SC as collateral and (ii) receive `loanAmt` units of a token from the SC.[1] At this point, client owes the bank `loanAmt` token units, which must be repaid by the specified `dueDate`. To this end, the client repeatedly invokes `repayInstallment(amt)`, where each invocation (i) authorizes the SC to transfer `amt` token units from the client to the bank, and (ii) accumulates the repaid amount in the variable `repaidAmt`. Prior to `dueDate`, the client may invoke `withdrawCollateral` to recover `collateralAmt` units of the collateral corresponding to

[1] We assume that the bank has previously sent at least `loanAmt` units of the token to the smart contract.

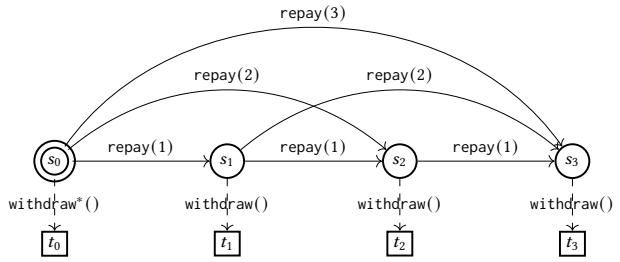


Figure 2: The state machine for the loan consists of the states s_i, the terminal states t_i, and the state transitions triggered by `repay` events. A `withdraw` event transforms a state into its corresponding terminal state. In addition, once the timeout expires, a `withdraw` event transforms s_0 into t_0.

`repaidAmt` repaid token units.[2] After `dueDate`, the bank can invoke `withdrawCollateral` to seize the entire collateral deposited in the SC. In this manner, the SC elegantly orchestrates the loan in token units secured by collateral in Ether, between the mutually distrustful parties (i.e., bank and client).

Limitations. The already projected growth of SC-based collateralized loans [26] can benefit from greater compatibility with existing blockchains by overcoming a fundamental limitation: SC-based collateralized loans can be deployed only on blockchains that support complex and stateful SC. As such, SC-based collateralized loans cannot be deployed in cryptocurrencies that either do not support any SC functionality, or are limited to stateless and relatively simple payment conditions (e.g., digital signature–based authorization), such as Bitcoin. Hence, users with holdings in Bitcoin cannot natively use them as collateral in SC-based collateralized loans, and are required to exchange them first for another cryptocurrency that supports SC.

Goal. In this work, we want to answer the following question: *Can we design a collateralized loan protocol that supports cryptocurrencies with limited SC capabilities?*

Towards our approach. We observe that the execution of a SC-based collateralized loan can be naturally unrolled as a *state machine* (SM), as illustrated in Figure 2. The SM is initialized in s_0: the initial state where the client has deposited the collateral and received the loan. The SM progresses according to two types of events. First, `repay(amt)` events (depicted by solid arrows) are triggered when the client repays an installment of `amt` token units. Each such event induces a transition from state s_i to s_j, where $j = i + \texttt{amt}$. Second, `withdraw()` events (depicted by dashed arrows) signal termination of the loan and effectively transition the SM from any state s_i to a terminal state t_i. Each terminal state t_i represents a final collateral distribution between bank and client in which a total of i coins have been repaid.

This SM abstraction decouples the loan logic from the underlying blockchains hosting the collateral and loan assets. Furthermore, although we use loans as the running example in this work, the same approach applies to a variety of other applications. In particular, we can derive a similar SM (although with different events triggering state transitions) for applications such as asset vesting, contractual payment obligations, and rental agreements, as we discuss in Section 5.4.

Our approach. We design a cryptographic protocol between the bank and client that serves as a flexible application layer on top of cryptocurrencies lacking support for complex, stateful SCs, thereby enabling the realization of the above SM-based collateralized loan. More concretely, assume a setting in which the collateral is held in Bitcoin and the loan assets are issued in BitcoinCash[3] (i.e., a cryptocurrency with the similar restricted SC capabilities as Bitcoin). The effect of each terminal state t_i in SM is realized by a Bitcoin transaction m_i that returns i units of Bitcoin to the client and the remaining $\texttt{collateralAmt} - i$ units to the bank. A `repay(amt)` event is triggered by the inclusion, in the BitcoinCash blockchain, of a transaction in which the client repays `amt` units of BitcoinCash to the bank. The cryptographic protocol that we design must enable the bank and the client to securely orchestrate the inclusion of the corresponding transactions in the Bitcoin and BitcoinCash blockchains, ensuring that the loan is settled correctly according to the SM logic, despite the mutually distrustful nature of the parties.

Challenges. There already exist cryptographic protocols that enable users to securely orchestrate the publication of transactions across different blockchains in support of applications such as atomic swaps [45], coin mixing [23, 27, 37, 43], off-chain payment channels [1, 2, 32], and zero-knowledge contingent payments [3, 7–9, 20, 25, 35, 44]. However, these protocols exhibit two key limitations. First, they typically enforce the synchronization of only a few pairs of transactions across two blockchains. For example, an atomic swap protocol ensures that a transaction on blockchain A is published if and only if a corresponding transaction on blockchain B is published. Second, they require both parties to remain online and interact throughout the protocol execution to monitor the blockchains and publish their transactions at the right time. In an atomic swap protocol, for instance, a party must include its transaction on blockchain B after observing that the counterparty has published its transaction on blockchain A.

These fundamental limitations prevent the direct application of existing protocols to our SM-based collateralized loan, which requires (i) the synchronization of multiple sequences of transactions across two blockchains (corresponding to the different combination of external events leading to an internal state, and the corresponding terminal state), and (ii) that after the SM is initialized in s_0, the bank can go offline and the client is the only one that remains online to enforce the loan settlement according to the SM logic, until the loan's due date.

1.1 Our Contributions

(1) New cryptographic primitive. To realize the SM-based logic, we introduce a new cryptographic primitive, called *verifiable graph encryption for signatures* (VGES). VGES allows to encrypt signatures on messages such that decryption is possible only in accordance with the graph-encoded logic of the SM. We leverage an oracle that produces an attestation for each transaction appended to the blockchain, representing a `repay(amt)` event. On an intuitive level,

[2] For ease of exposition, we assume a 1:1 equivalence between repayments and collateral.

[3] BitcoinCash is a fork of Bitcoin, and it ranks among the top twelve cryptocurrencies by market capitalization [10].

for each terminal state t_i, the bank generates a signature σ_i authorizing the transaction m_i that distributes the collateral between the bank and the client. The bank then encrypts these signatures using VGES and provides the resulting ciphertext to the client. As the client publishes transactions corresponding to `repay(amt)` events, they obtain the corresponding attestations from the oracle, allowing progression in the SM. Upon reaching a terminal state t_i, the client uses the attestations obtained from the oracle to decrypt σ_i from the VGES ciphertext, and subsequently use it to authorize the transaction m_i that settles the loan according to t_i. A VGES satisfies *graph enforcement* if the client can obtain σ_i only by triggering events that enable reaching t_i in the SM. Additionally, VGES satisfies *verifiability* if it allows the client, prior to initiating the loan, to verify that the ciphertext enables recovery of the signatures corresponding to all terminal states of the SM.

(2) Provably secure constructions. We propose two constructions for VGES and formally prove that they satisfy *graph enforcement* and *verifiability*. The constructions differ in the number of `repay(amt)` events included in the SM and, consequently, in the number of on-chain repayment transactions required to settle the loan. The first construction encodes the SM as a *transitive tournament* graph. Concretely, consider a graph where each vertex v_i represents state s_i and its corresponding terminal state t_i via transaction m_i, with directed edges denoting `repay(amt)` events that transition from one state to another. In a transitive tournament graph, every pair of nodes is connected by a directed edge; that is, for all $j < i$, a `repay(amt)` event transitions from s_j to s_i. This construction permits a more flexible loan repayment, as the client can repay any amount at any time. However, this construction incurs higher off-chain costs to compute and verify the VGES ciphertext. The second construction encodes the SM as a *linear graph*. Concretely, `repay(amt)` events transition between consecutive states, i.e., from s_i to s_{i+1}. This construction permits a more restricted loan repayment, where the client must repay the loan in fixed-size installments. However, this construction incurs lower off-chain costs. We thereby show that VGES is the flexible cryptographic cornerstone to build cryptographic collateralized loans without smart contracts over blockchains that support digital signatures for authorizing basic transactions, hence realizing the loan application backwards-compatible with many of the existing blockchains, including Bitcoin.

(3) Implementation. We provide a Rust-based implementation for each construction, leveraging the `bls12_381` and `secp256kfun` crates for pairing and elliptic curve operations. Our implementation incorporates performance optimizations, including precomputation and parallelization, which we describe in Section 4.3. We evaluate the on-chain costs of both constructions, as well as off-chain costs as the number N of SM states increases. Our benchmarks show that the construction for the linear graph is approximately 10× faster than the construction for the transitive tournament graph, but incurs up to N times more on-chain transactions in the worst case. Overall, our performance evaluation demonstrates that both constructions are practical on commodity hardware.

2 Technical Overview

In this section we provide a technical overview of VGES, the core cryptographic building block that allows us to realize the collateralized loan without smart contracts. Then, we summarize in Figure 4 how we use VGES for cryptographic collateralized loans.

2.1 States and Events in the Blockchains

We first provide an overview of our system model and how we represent terminal states and events in the state machine (SM) as blockchain transactions.

System model. We consider two blockchains, denoted as $\mathcal{L}$ and $\widehat{\mathcal{L}}$,[4] modeled as an append-only list of transactions. To submit a transaction to blockchain $\mathcal{L}$, the sender must include a signature computed as $\sigma \leftarrow \mathsf{Sig}(sk_S, m)$. Upon receiving a transaction request (m, σ), the blockchain is trusted such that only valid transactions are added to the blockchain (safety) and any valid transaction is eventually added to the blockchain (liveness). The model is analogous for blockchain $\widehat{\mathcal{L}}$. We hereby assume w.l.o.g. that $\mathcal{L}$ is used to handle the collateral and $\widehat{\mathcal{L}}$ is were the client borrows and repays the loan coins.

The client and the bank. The client is identified by the key pair (pk_C, sk_C) in $\mathcal{L}$ and the key pair $(\widehat{pk}_C, \widehat{sk}_C)$ in $\widehat{\mathcal{L}}$. Similarly, the bank is identified by the key pair $(pk_{\mathcal{B}}, sk_{\mathcal{B}})$ in $\mathcal{L}$ and the key pair $(\widehat{pk}_{\mathcal{B}}, \widehat{sk}_{\mathcal{B}})$ in $\widehat{\mathcal{L}}$. The bank and the client can additionally create an escrow address in $\mathcal{L}$, denoted as $\mathsf{esc} := (pk_{\mathcal{B}}, pk_C)$. Submitting transaction m_i spending from esc requires both (i) the authorization $\sigma_{\mathcal{B},i}$ for m_i done with $sk_{\mathcal{B}}$; and (ii) the authorization $\sigma_{C,i}$ for m_i done with sk_C.

Mapping states to transactions in $\mathcal{L}$. We implement the initial state s_0 as an escrow address esc in $\mathcal{L}$ that holds the collateral deposited by the client. An escrow address requires joint authorization, i.e., signatures from both the bank and the client for any transaction, hence the bank is ensured that the client can only release the collateral by means of transactions authorized by the bank. A common fail-safe mechanism for escrow addresses is that funds are returned to their owner after a predefined timeout T (i.e., variable `dueDate` in Figure 1). Although this mechanism is often implemented via scripting in existing blockchains, it can also be realized without scripts using cryptographic primitives [45]. In our case, although the client is the owner of the collateral, the fail-safe returns the coins to the bank if T expires. Therefore, the client must repay the loan to reclaim the collateral.

Recall that upon reaching a terminal state t_i in the SM, the collateral is distributed between the bank and the client in proportion to the amount of the loan that has been repaid. In blockchain $\mathcal{L}$, this outcome is represented by mapping each terminal state t_i to a simple transaction m_i that spends from the escrow address esc and sends i units of collateral to the client and the rest to the bank. Finally, intermediate states s_i are logical states that are not explicitly represented by some transaction in $\mathcal{L}$, and are instead enforced by our VGES cryptographic construction.

Mapping repay and withdraw events. Recall that each `repay(amt)` event is triggered when the client repays a loan installment of amount `amt`. In our implementation, such an event occurs when

[4] For readability, we denote with $\widehat{\ }$ anything related to blockchain $\widehat{\mathcal{L}}$.

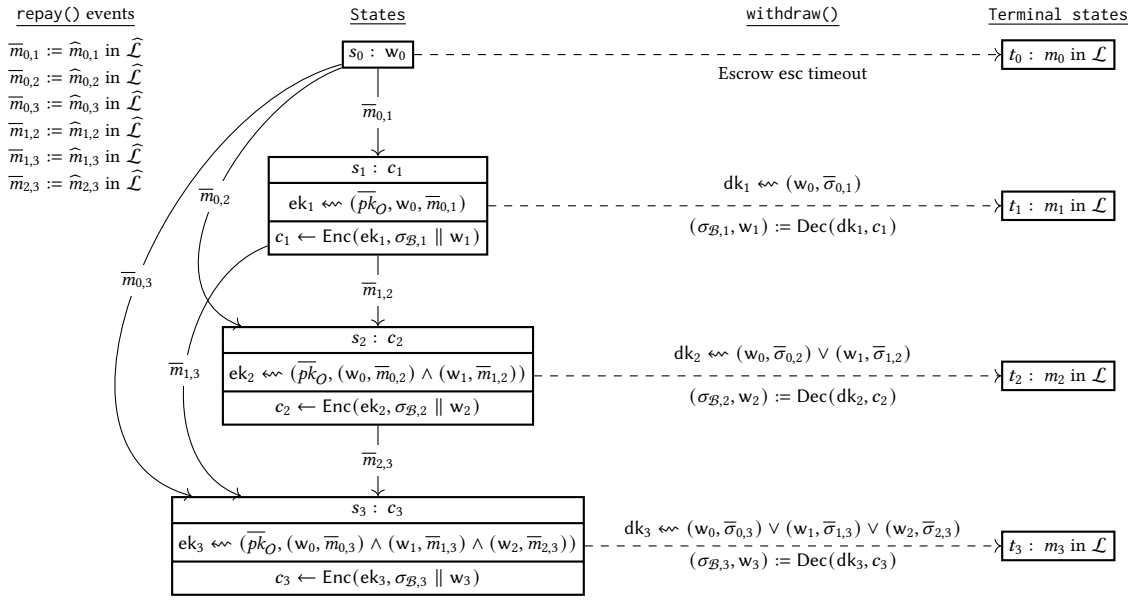


Figure 3: Overview of how the state machine in Figure 2 is implemented using a cryptographic protocol. Each `repay` action corresponds to an event $\overline{m}_{j,i}$ indicating that a transaction $\widehat{m}_{j,i}$ is recorded in $\widehat{\mathcal{L}}$. Each state is represented by a ciphertext c_i, except for the initial state s_0, which is represented by the initial witness w_0. Each `withdraw` operation decrypts the corresponding ciphertext c_i, except for the `withdraw` of the initial state s_0, which occurs upon escrow timeout. Finally, each terminal state corresponds to a transaction m_i in $\mathcal{L}$ that distributes the collateral between the bank and the client.

the client includes a transaction $\widehat{m}$ in $\widehat{\mathcal{L}}$ that transfers `amt` token units to the bank. In contrast, `withdraw()` events are purely logical events (i.e., transition from s_i to t_i), that are enabled by our VGES cryptographic construction.

2.2 Cryptographic Building Block: VGES

We now present our cryptographic construction that enforces the SM logic without relying on smart contracts. Our goal is to build a ciphertext c that ensures the following: the client can obtain the authorization $\sigma_{\mathcal{B},t}$ for the transaction m_t corresponding to a target terminal state t_t *if and only if* they have already submitted to $\widehat{\mathcal{L}}$ all the required transactions $\widehat{m}_{j,i}$ dictated by the SM.

To realize this, we face two main challenges. The first challenge is how to encrypt a signature such that it can only be decrypted if a transaction is included in the blockchain (i.e. a single transition in the graph). The second challenge is incremental on the first: how to ensure that the ciphertext is decrypted only if a concrete sequence of transactions is in the blockchain (i.e. multiple transitions in the graph). We next discuss these challenges and provide an intuition of our cryptographic primitive. Finally, we describe how to use the primitive to instantiate a collateralized loan.

Single transition. To build intuition for our construction, we first consider the realization of a single transition from state s_j to state s_i with $j < i$ in the SM. The bank encrypts the authorization $\sigma_{\mathcal{B},i}$ for transaction m_i so that the client can decrypt it only after including in $\widehat{\mathcal{L}}$ the transaction $\widehat{m}_{j,i}$ corresponding to the `repay(amt)` event from state s_j to state s_i. This enforces the required dependency: m_i can appear in $\mathcal{L}$ if and only if $\widehat{m}_{j,i}$ has previously appeared in $\widehat{\mathcal{L}}$.

Intuitively, this can be realized using a witness encryption scheme that allows the bank to encrypt $\sigma_{\mathcal{B},i}$ under the statement that says "$\widehat{m}_{j,i}$ is in $\widehat{\mathcal{L}}$". The client can then decrypt $\sigma_{\mathcal{B},i}$ only after publishing $\widehat{m}_{j,i}$ in $\widehat{\mathcal{L}}$, since only then would they obtain the corresponding witness. While there are not efficient implementations of witness encryption for arbitrary NP relations [21], prior works [9, 15, 31] have shown efficient constructions for the concrete case with a statement that says "provide a valid signature for $\widehat{m}_{j,i}$" and the witness is such a signature. Notably, we cannot rely on the client's signature on $\widehat{m}_{j,i}$, since the client could generate it locally without actually including $\widehat{m}_{j,i}$ in $\widehat{\mathcal{L}}$, thus breaking the desired dependency.

Instead, we rely on a cryptographic oracle, represented by the tuple $(\overline{pk}_O, \overline{sk}_O)$. As used in prior works [9, 15, 31] the oracle is our solution to the challenge of obtaining a signature for the message $\overline{m}_{j,i}$:="transaction $\widehat{m}_{j,i}$ is in $\widehat{\mathcal{L}}$". To do so, the oracle produces an attestation for each transaction that has been appended to the blockchain $\widehat{\mathcal{L}}$. An attestation is a tuple of message $\overline{m}$ and signature $\overline{\sigma}$, that, if it verifies for $\overline{pk}_O$ then is considered as a proof that transaction $\widehat{m}$ is indeed in blockchain $\widehat{\mathcal{L}}$.[5] Both the client and the bank have unrestricted read access to the signatures produced by the oracle. In practice, a similar functionality is already realized in some blockchains like Ethereum, where validators sign the transactions included in the blocks they produce. To further distribute the trust,

[5]For efficiency, the oracle may sign only the receiver key and amount of $\widehat{m}$ as a the message $\overline{m}$, rather than signing the entire transaction. This is enough for applications that rely on knowing the balance of a certain user (e.g., the token units returned to the keys of the bank).

it is possible to rely on multiple independent oracles, as in [31]. We discuss this extension in Section 5.2.

Multiple transitions. As shown in Figure 2, a single state s_i in the SM may be reached through multiple paths. Given the infrastructure described above for implementing a single transition, a naive approach to supporting multiple transitions would treat each single-transition mechanism as a black box and encrypt the authorization $\sigma_{\mathcal{B},i}$ under a statement such as: "$\{\widehat{m}_{0,t}$ in $\widehat{\mathcal{L}}\}$ or $\{\widehat{m}_{0,1}$ and $\widehat{m}_{1,t}$ are in $\widehat{\mathcal{L}}\}$ or $\{\widehat{m}_{0,1}$ and $\widehat{m}_{1,2}$ and $\widehat{m}_{2,t}$ are in $\widehat{\mathcal{L}}\}$ or ... or $\{\widehat{m}_{0,1}$ and $\widehat{m}_{1,2}$ and ... and $\widehat{m}_{t\text{-}1,t}$ are in $\widehat{\mathcal{L}}\}$", including all possible paths. However, existing constructions for signature-based witness encryption [9, 15, 31] do not support this functionality. They can compose OR statements (i.e., produce one ciphertext per OR-branch encrypting the same signature $\sigma_{\mathcal{B},i}$), but they cannot natively compose AND statements. For instance, client may require that the oracle attests "$\{\widehat{m}_{0,1}$ and $\widehat{m}_{1,t}$ are in $\widehat{\mathcal{L}}\}$". However, the oracle cannot provide such attestations without knowing the full logic of the loan, including all possible sequences of repayment transactions. Furthermore, each bank and client would need to interact with the oracle during loan setup and reveal the logic of the loan, and the oracle would have to actively monitor $\widehat{\mathcal{L}}$ for all possible transaction sequences corresponding to every active loan. Therefore, this approach is impractical. The key contribution of our approach beyond existing work is that our primitive supports the composition of both OR and AND statements without increasing interaction with the oracle. We illustrate our approach in Figure 3. First, the bank samples a fresh witness–statement pair $(\mathsf{w}_i, \mathsf{x}_i)$ for each state s_i and encrypts the signature $\sigma_{\mathcal{B},i}$ (on the message m_i) together with the witness w_i under the key ek_i, producing the ciphertext c_i.

To decrypt a ciphertext c_i, the client must derive the corresponding key dk_i. This key is computed using the witness w_j of a preceding state and the attestation $\overline{\sigma}_{j,i}$ for the event $\overline{m}_{j,i}$ that authorizes the transition from s_j to s_i. For example, dk_2 may be derived from $(\mathsf{w}_0, \overline{\sigma}_{0,2})$ or from $(\mathsf{w}_1, \overline{\sigma}_{1,2})$. Since w_0 is initially given to the client, the pair $(\mathsf{w}_0, \overline{\sigma}_{0,2})$ immediately enables the derivation of dk_2 as it is a direct transition from the initial state. In contrast, deriving dk_2 from $(\mathsf{w}_1, \overline{\sigma}_{1,2})$ requires the client to have already decrypted c_1 to obtain w_1, which in turn requires the tuple $(\mathsf{w}_0, \overline{\sigma}_{0,1})$. Thus, the client is effectively forced to collect all witnesses w_i corresponding to the events needed to reach the target state s_t, i.e., satisfying one of the AND sub-statements in "$\{\widehat{m}_{0,t}$ in $\widehat{\mathcal{L}}\}$ or $\{\widehat{m}_{0,1}$ and $\widehat{m}_{1,t}$ are in $\widehat{\mathcal{L}}\}$ or $\{\widehat{m}_{0,1}$ and $\widehat{m}_{1,2}$ and $\widehat{m}_{2,t}$ are in $\widehat{\mathcal{L}}\}$ or ... or $\{\widehat{m}_{0,1}$ and $\widehat{m}_{1,2}$ and ... and $\widehat{m}_{t\text{-}1,t}$ are in $\widehat{\mathcal{L}}\}$".

Proving that c is well-formed. The bank employs a non-interactive zero knowledge proof (NIZK). The NIZK proves that each ciphertext c_i: (i) contains an authorization that verifies for $pk_{\mathcal{B}}$ and m_i; (ii) contains the witness w_i corresponding to the statement x_i; and (iii) can be decrypted with the witness w_j corresponding to the statement x_j and an attestation valid for $\overline{pk}_{\mathcal{O}}$ and repayment option $\overline{m}_{j,i}$ for every $j < i$. These three conditions assure the client that each c_i is well-formed and that they can extract a signature $\sigma_{\mathcal{B},i}$ if they submit repayment transactions to $\widehat{\mathcal{L}}$. The bank sets the c as a tuple comprising all ciphertexts c_i, statements x_i, the witness w_0, and the NIZK discussed above and sends it to the client.

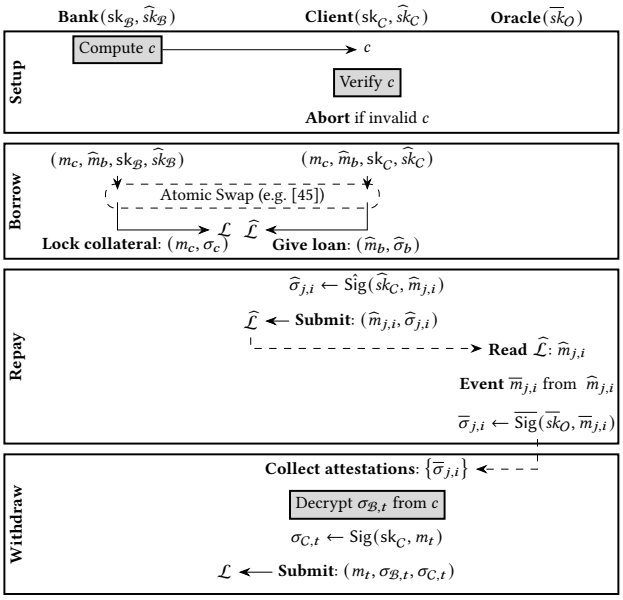


Figure 4: Cryptographic collateralized loan runs in four stages. During setup, bank generates c and provides it to client, who verifies it. Upon successful verification, during borrow, bank and client execute an atomic swap that locks the collateral in $\mathcal{L}$ (m_c) and transfers the loan to client in $\widehat{\mathcal{L}}$ ($\widehat{m}_b$). During repay, client submits repayment transactions, while the oracle attests to transactions appended to $\widehat{\mathcal{L}}$; client collects attestations to the target terminal state t_t, represented by transaction m_t. Finally, during withdraw, client uses these attestations to decrypt the signature $\sigma_{\mathcal{B},t}$ from c. Gray boxes indicate the steps computed using VGES.

Security intuition. Regarding the enforcement of the SM logic, consider a client attempting to obtain the signature for m_2 by submitting transaction $\widehat{m}_{1,2}$ to $\widehat{\mathcal{L}}$, without submitting $\widehat{m}_{0,1}$ (see Figure 3). In this case, the only attestation they would collect is $\overline{\sigma}_{1,2}$. However, dk_2 can only be derived from one of the following tuples: $(\mathsf{w}_0, \overline{\sigma}_{0,2})$ or $(\mathsf{w}_1, \overline{\sigma}_{1,2})$. Since the adversary does not know w_1, because they have not collected $\overline{\sigma}_{0,1}$, they cannot derive dk_2. Regarding verifiability, consider a bank attempting to provide a malformed c that still passes verification. For this to occur, the bank needs to break the soundness of the NIZK.

Collateralized loan: protocol overview. The loan protocol follows as described in Figure 4. First, the bank computes ciphertext c, where the bank encrypted the authorizations $\sigma_{\mathcal{B},i}$ for all the transactions m_i corresponding to terminal states t_i in the SM. The client receives c, who verifies that it is well-formed. To enter the initial state of the SM, the client authorizes a transaction m_c in $\mathcal{L}$ that deposits the collateral into the escrow address esc, while the bank authorizes a transaction $\widehat{m}_b$ in $\widehat{\mathcal{L}}$ that transfers the loan to the client. Since the incentives are misaligned at this stage, we rely on an atomic swap protocol (e.g., [45]) to execute transactions m_c and $\widehat{m}_b$ atomically, ensuring that either both transactions are confirmed or neither is. Upon successful completion of the atomic

swap, the initial state has been reached and the client can decrypt from c the authorization $\sigma_{\mathcal{B},t}$ for transaction m_t corresponding to terminal state t_t after submitting to $\widehat{\mathcal{L}}$ all the required transactions $\widehat{m}_{j,i}$ according to the SM. For ease of exposition, the protocol we describe assumes a static exchange rate between the collateral and the loan, allowing us to highlight where our primitive VGES is used. In Appendix D we discuss how to modify the protocol in Figure 4 to enable dynamic exchange rate between ledgers. To do so, we define mechanisms for liquidation and margin calls, commonly used in lending. Regarding privacy, it is worth noting that although basic transactions reveal less application-level information than smart contracts, they can still leak meaningful insights. For example, in the protocol of Figure 4, blockchain observers may be able to correlate repayment transactions on ledger $\widehat{\mathcal{L}}$ (repayment phase) with transactions on $\mathcal{L}$ during the borrowing and withdrawal phases, thereby inferring that a loan has taken place. In Appendix D, we discuss several techniques to address this.

3 Preliminaries

Notation. Assign ($:=$) stores values from a deterministic operation, gets ($\leftarrow$) stores values from a probabilistic operation, parse ($\leftarrow$) parses data from a variable and ($\xleftarrow{\$}$) denotes sampling an element uniformly at random from a finite set. We denote running n executions of algorithm f as f^n.

Relation. Let $\mathcal{R} \subseteq \mathcal{D}_S \times \mathcal{D}_w$ be a relation with statement-witness pairs $(\mathsf{x}, \mathsf{w}) \in \mathcal{D}_S \times \mathcal{D}_w$. We denote by $\mathcal{L}_\mathcal{R}$ the associated language defined as $\mathcal{L}_\mathcal{R} := \{\mathsf{x} \in \mathcal{D}_S \mid \exists \mathsf{w} \in \mathcal{D}_w \text{ s.t. } (\mathsf{x}, \mathsf{w}) \in \mathcal{R}\}$. We require the following properties: (i) There exists a PPT algorithm $\mathsf{createR}(1^\lambda)$ that computes $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$. Note that this implies that $|\mathsf{x}|, |\mathsf{w}| \leq \mathsf{poly}(\lambda)$; and (ii) the relation is decidable in polynomial time. A relation is *hard* if for all PPT adversaries $\mathcal{A}$ the probability that on input x, $\mathcal{A}$ outputs w such that $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$ is negligible.

Digital signature scheme. Let DS:=(KGen, Sig, Vf) be a digital signature scheme where: (i) KGen gets as input 1^λ and outputs key pair (pk, sk); (ii) Sign gets as input sk and message m, outputs signature σ; and (iii) Vf gets as input pk, message m, and signature σ, outputs a bit b. A DS is correct if for all messages m, it holds that $\mathsf{Vf}(pk, m, \mathsf{Sig}(sk, m)) = 1$ and secure under EUF-CMA [24].

Adaptor signature scheme. Let ADP:=(PreSig, PreVf, Adapt, Extract) be an adaptor signature scheme [1, 11, 22] defined with respect to a digital signature scheme DS and a relation $\mathcal{R}$ where: (i) PPT algorithm PreSig gets as input signing key sk, message m, and statement x, and outputs pre-signature $\widetilde{\sigma}$; (ii) DPT algorithm PreVf gets as input verification key pk, message m, statement x and pre-signature $\widetilde{\sigma}$ and outputs a bit b; (iii) DPT algorithm Adapt gets as input pre-signature $\widetilde{\sigma}$ and witness w, and outputs signature σ; and (iv) DPT algorithm Extract gets as input signature σ, pre-signature $\widetilde{\sigma}$ and statement x, and outputs witness w. We require ADP to be correct and secure for full extractability, adaptability and pre-verify soundness, as defined in [11, 22].

Witness encryption based on signatures. Let WES := (Enc, Dec) be a witness encryption based on signatures [31], defined with respect to a digital signature scheme $\overline{\mathsf{DS}} = (\overline{\mathsf{KGen}}, \overline{\mathsf{Sig}}, \overline{\mathsf{Vf}})$, where: (i) PPT algorithm Enc gets as input a verification key $\overline{pk}$ and a message $\overline{m}$, a plaintext m, and outputs a ciphertext c; and (ii) DPT algorithm Dec gets as input a signature $\hat{\sigma}$ and a ciphertext c, and outputs a plaintext m. WES is correct and secure for IND-CPA [31].

4 Verifiable Graph Encryption for Signatures

In this section we define our new primitive *verifiable graph encryption for signatures* (VGES). We then present two constructions of VGES and evaluate their performance. The VGES formalizes the functionality of the state machine (SM) described in Section 1.

Recall that the SM described in Sections 1 and 2 consists of states s_i, terminal states t_i (each associated with a transaction m_i), and events that enable transitions between states, denoted by $\overline{m}_{j,i}$. To simplify the notation, we observe from Figures 2 and 3 that the logic of the SM can be condensed in a graph. Concretely, if we have a graph $G := (V, E)$, each vertex v_i of the graph corresponds to a state s_i. Similarly, we associate to the same vertex v_i a terminal state t_i, which is represented by transaction m_i. On the other hand, each edge $(v_j, v_i) \in E$ corresponds to an event $\overline{m}_{j,i}$. The SM is represented by the family of graphs that satisfy Definition 1 below.

Definition 1 (Graphs supported by VGES)**.** *Let $G := (V, E)$ be a directed acyclic graph (DAG), and let $v_0 \in V$ be a vertex of G. We say that v_0 is a* from-root *(or simply a* root*) of G if, for every other vertex $v_i \in V$, there exists a directed path E' from v_0 to v_i. We say that the primitive* VGES supports *G if G is a directed acyclic graph and has exactly one vertex that is a from-root of G.*

We define VGES with regards to a graph G to simplify the notation. However, it does not encrypt a graph. Instead, it encrypts a list of signatures on transactions m_i that can only be decrypted once the required attestations on events $\overline{m}_{j,i}$ have been collected, following the dependencies encoded by the graph G (cf. Figure 3).

Definition 2 (Verifiable Graph Encryption for Signatures)**.** *Let* $\mathsf{DS} := (\mathsf{KGen}, \mathsf{Sig}, \mathsf{Vf})$ *and* $\overline{\mathsf{DS}} := (\overline{\mathsf{KGen}}, \overline{\mathsf{Sig}}, \overline{\mathsf{Vf}})$ *be digital signatures schemes. Let $G := (V, E)$ satisfy Definition 1. A* VGES *scheme with respect to* DS *and* $\overline{\mathsf{DS}}$ *consists on the PPT algorithms* $\mathsf{VGES} = (\mathsf{EncGS}, \mathsf{VfEncGS}, \mathsf{DecGS})$ *with the following syntax:*

- $c \leftarrow \mathsf{EncGS}(sk, \overline{pk}, G)$*: PPT algorithm that takes as input the secret key sk, the public key* $\overline{pk}$ *and the graph G. It outputs a ciphertext c.*
- $b := \mathsf{VfEncGS}(c, pk, \overline{pk}, G)$*: DPT algorithm that takes as input a ciphertext c, the public key pk, the public key* $\overline{pk}$ *and the graph G. It outputs a bit b.*
- $\sigma_t := \mathsf{DecGS}(c, \{\overline{\sigma}_{j,i}\}_{E'})$*: DPT algorithm that takes as input ciphertext c, and a set of signatures* $\{\overline{\sigma}_{j,i}\}_{E'}$ *for each edge* $\overline{m}_{j,i}$ *in the subset of edges $E' \subset E$ that form a path from the root to vertex v_t, associated to transaction m_t. It outputs signature σ_t.*

Definition 3 (Correctness)**.** *We say that a* VGES *scheme is* correct *if for all $\lambda \in \mathbb{N}$, all graphs $G := (V, E)$ satisfying Definition 1, all paths E' from the root to any other vertex v_t associated with message m_t, in the graph G, and the corresponding sets of signatures* $\{\overline{\sigma}_{j,i}\}_{E'}$

for each path E', the following holds:

$$\Pr\left[\begin{array}{c|l} & (\overline{pk},\overline{sk}) \leftarrow \overline{\mathsf{DS}}.\overline{\mathsf{KGen}}(1^\lambda) \\ & (pk,sk) \leftarrow \mathsf{DS}.\mathsf{KGen}(1^\lambda) \\ b_0 = 1 & c \leftarrow \mathsf{EncGS}(sk,\overline{pk},G) \\ \wedge\, b_1 = 1 & \sigma_t := \mathsf{DecGS}(c,\{\overline{\sigma}_{j,i}\}_{E'}) \\ \wedge\, b_2 = 1 & b_0 := \mathsf{VfEncGS}(c,pk,\overline{pk},G) \\ & b_1 := \forall\ \overline{m}_{j,i} \in E',\ \overline{\mathsf{Vf}}(\overline{pk},\overline{m}_{j,i},\overline{\sigma}_{j,i}) = 1 \\ & b_2 := \mathsf{Vf}(pk,m_t,\sigma_t) = 1 \end{array}\right] = 1.$$

Definition 4 (Graph enforcement)**.** *Let* $\mathsf{DS} := (\mathsf{KGen},\mathsf{Sig},\mathsf{Vf})$ *and* $\overline{\mathsf{DS}} := (\overline{\mathsf{KGen}},\overline{\mathsf{Sig}},\overline{\mathsf{Vf}})$ *be digital signatures schemes. Let* $G := (V,E)$ *satisfy Definition 1. Let* $\mathsf{VGES} = (\mathsf{EncGS},\mathsf{VfEncGS},\mathsf{DecGS})$ *be a* VGES *scheme with respect to* DS, $\overline{\mathsf{DS}}$*. We say that* VGES *satisfies graph enforcement, if for every PPT adversary* $\mathcal{A}$*, the probability that the following game outputs* 1 *is negligible:*

1. ***Key generation.*** *Generate keys* $(\overline{pk},\overline{sk}) \leftarrow \overline{\mathsf{DS}}.\overline{\mathsf{KGen}}(1^\lambda)$ *and* $(pk,sk) \leftarrow \mathsf{DS}.\mathsf{KGen}(1^\lambda)$.
2. ***Signing oracles.*** *Let* $O\mathsf{Sig}$ *and* $O\overline{\mathsf{Sig}}$ *be two signing oracles, i.e., on input* m*, return* $\sigma \leftarrow \mathsf{DS}.\mathsf{Sig}(sk,m)$ *and on input* $\overline{m}$*, return* $\overline{\sigma} \leftarrow \overline{\mathsf{DS}}.\overline{\mathsf{Sig}}(\overline{sk},\overline{m})$.
3. ***Preprocessing phase.*** *Run* $\mathcal{A}$ *with access to* $O\mathsf{Sig}$ *and* $O\overline{\mathsf{Sig}}$ *on input* $(pk,\overline{pk})$*. Receive graph* $G := (V,E)$*, where* $|V| = N+1$ *(i.e.* N *vertices and the root), from* $\mathcal{A}$.
4. ***Ciphertext generation.*** *Compute* EncGS *on input* sk, $\overline{pk}$, G*, to produce* c.
5. ***Online phase.*** *Run* $\mathcal{A}$ *with access to* $O\mathsf{Sig}$ *and* $O\overline{\mathsf{Sig}}$ *on input* c*. Obtain a tuple* (σ^*, m_t) *with* $0 < t \le N$ *from* $\mathcal{A}$.
6. ***Winning conditions.*** *Output* 1 *if the following hold:*
   - ***Non-triviality:*** *The message* m_t *corresponding to vertex* $v_t \in V$ *has never been queried to* $O\mathsf{Sig}$.
   - ***Skipped-attestation:*** *The messages* $\overline{m}_{j,i}$ *queried to* $O\overline{\mathsf{Sig}}$ *correspond to a subset of edges,* $E' \subset E$*, such that* E' *does not contain a path from* v_0 *to* v_t.
   - ***Valid forgery:*** $\mathsf{Vf}(pk,m_t,\sigma^*) = 1$.

Definition 5 (Verifiability)**.** *Let* $\mathsf{DS} := (\mathsf{KGen},\mathsf{Sig},\mathsf{Vf})$ *and* $\overline{\mathsf{DS}} := (\overline{\mathsf{KGen}},\overline{\mathsf{Sig}},\overline{\mathsf{Vf}})$ *be digital signatures schemes. Let* $G := (V,E)$ *satisfy Definition 1. Let* $\mathsf{VGES} = (\mathsf{EncGS},\mathsf{VfEncGS},\mathsf{DecGS})$ *be a* VGES *scheme with respect to* DS *and* $\overline{\mathsf{DS}}$*. We say that* VGES *satisfies verifiability, if for every PPT adversary* $\mathcal{A}$*, the probability that the following game outputs* 1 *is negligible:*

1. ***Initiate adversary.*** *Run* $\mathcal{A}$ *on input* λ *and obtain a graph* $G := (V,E)$*, keys* pk *and* $\overline{pk}$*, a ciphertext* c *and index* t *in return.*
2. ***Receive attestations.*** *Select a subset of edges* $E' \subset E$ *forming a path from* v_0 *to* v_t*. Run* $\mathcal{A}$ *on input* E' *to receive an attestation* $\overline{\sigma}_{j,i}$ *for each* $\overline{m}_{j,i}$ *corresponding to each edge in* E'.
3. ***Winning condition.*** *Output* 1 *if the following holds:*
   - ***Valid ciphertext:*** $\mathsf{VfEncGS}(c,pk,\overline{pk},G) = 1$.
   - ***Valid attestation:*** *For each message* $\overline{m}_{j,i}$ *and* $\overline{\sigma}_{j,i}$*, it holds that* $\overline{\mathsf{DS}}.\overline{\mathsf{Vf}}(pk,\overline{m}_{j,i},\overline{\sigma}_{j,i}) = 1$.
   - ***Failed decryption:*** *Compute* $\sigma_t := \mathsf{DecGS}(c,\{\overline{\sigma}_{j,i}\}_{E'})$ *such that* $\mathsf{Vf}(pk,m_t,\sigma_t) = 0$.

4.1 Construction for a Transitive Tournament

The first construction we discuss considers as input a transitive tournament graph, which we depict in Figure 5. It is easy to see that

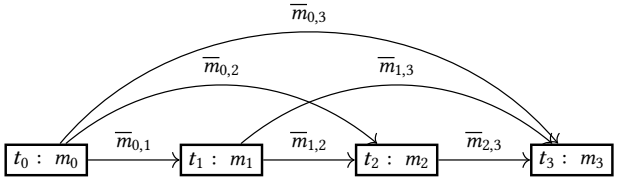


Figure 5: Example of a transitive tournament graph that compresses the SM in Figure 3.

a transitive tournament satisfies Definition 1, since it is a directed acyclic graph that has a single from-root vertex from which there is a path to any other vertex in the graph. Interestingly, we already used a transitive tournament to depict the logic of the loan contract in Figure 2, because it captures the logic of a loan in which a client can repay the loan in multiple different ways. We present below the key intuitions of how our construction works, and refer to Figure 6 for the full details of the construction.

Intuitions for encryption. Recall that in Figure 3 we showed two key ideas regarding encryption key ek_i: (i) we encrypt both σ_i and w_i under the same key ek_i; and (ii) key dk_i can be derived from the preceding witness w_j and the attestation on event $\overline{m}_{j,i}$. In our construction, the analogue of ek_i is $\tilde{y}_{j,i}$. To encrypt σ_i, we use adaptor signatures with $\tilde{Y}_{j,i} := g^{\tilde{y}_{j,i}}$ as the public statement. The resulting presignature $\widetilde{\sigma}_{j,i} \leftarrow \mathsf{ADP.PreSig}(sk,m_i,\tilde{Y}_{j,i})$ acts as a ciphertext for a signature σ_i on m_i. To encrypt the witness w_i, we compute $\mathsf{ct}_{j,i}$ as the one-time pad of w_i with $\tilde{y}_{j,i}$. Thus, both the signature and witness are encrypted under the same key (i.e., $\tilde{y}_{j,i}$). To derive $\tilde{y}_{j,i}$ from the preceding witness w_j and an attestation on the event $\overline{m}_{j,i}$, we construct $\tilde{y}_{j,i} := \mathsf{w}_j + y_{j,i}$, where w_j is the witness corresponding to the predecessor vertex m_j and $y_{j,i}$ is a random fresh element. Finally, to tie $y_{j,i}$ to the attestation on the event $\overline{m}_{j,i}$ we could simply $c_{j,i} \leftarrow \mathsf{WES.Enc}((\overline{pk},\overline{m}_{j,i}),y_{j,i})$, ensuring that $y_{j,i}$ can only be decrypted with $\overline{\sigma}_{j,i}$. In the construction, we take a slightly more complex approach for this last step, which is required for the cut-and-choose proof as we explain below, in the *intuitions for verification* paragraph.

Intuitions for decryption. As shown in Figure 3, to decrypt w_i and σ_i, a party needs both the attestation $\overline{\sigma}_{j,i}$ and the preceding witness w_j to compute the decryption key dk_i, which corresponds to $\tilde{y}_{j,i}$ in our construction. For that, they first decrypt $c_{j,i}$ to obtain $y_{j,i}$, and compute $\tilde{y}_{j,i} = \mathsf{w}_j + y_{j,i}$. Then, they use $\tilde{y}_{j,i}$ to (i) decrypt $\mathsf{ct}_{j,i}$ and obtain w_i; and (ii) adapt (i.e., decrypt) the pre-signature $\widetilde{\sigma}_{j,i}$ and thereby obtain σ_i.

Intuitions for verification. We use a cut-and-choose to construct the proof that the generated ciphertexts are well-formed. All components except for the witness encryption ciphertext $c^{(k)}_{j,i}$ can be publicly verified: adaptor signatures have a verification algorithm, and one-time pads can be checked using public information. To demonstrate that $c^{(k)}_{j,i}$ is well-formed we rely on cut-and-choose to construct the proof. So, instead of simply encrypting $y_{j,i}$ as $\mathsf{WES.Enc}((\overline{pk},\overline{m}_{j,i}),y_{j,i})$ we follow the cut-and-choose strategy as follows. For each tuple $(\overline{pk},\overline{m}_{j,i})$, the encryptor produces γ ciphertexts $c^{(k)}_{j,i}$, each encrypting a distinct value $r^{(k)}_{j,i}$. After generating all

ciphertexts, a random subset ($\approx \frac{\gamma}{2}$) is selected for opening. For each opened ciphertext, the encryptor reveals $r_{j,i}^{(k)}$ and the randomness used, allowing anyone to recompute and verify $c_{j,i}^{(k)}$. The unopened ciphertexts are then accompanied by the value $s_{j,i}^{(k)} := r_{j,i}^{(k)} + y_{j,i}$. This technique assign $s_{j,i}^{(k)}$ values only to unopened ciphertexts because if it would use any of the opened $r_{j,i}^{(k)}$, the receiver would trivially learn all $y_{j,i}$. The core security guarantee is that, because the encryptor does not know in advance which ciphertexts will be opened, they cannot cheat selectively. If all opened ciphertexts are valid (i.e., encrypted the expected $r_{j,i}^{(k)}$), then with overwhelming statistical probability at least one unopened ciphertext is also valid. Therefore, when the decryptor decrypts all the unopened ciphertexts, they can be confident that at least one of them will decrypt to a value $r_{j,i}^{(k)}$ that can be used to obtain $y_{j,i}$ from the corresponding $s_{j,i}^{(k)}$.

How the construction captures the graph. So far, we have provided intuition for how encryption, decryption and verification operate. We now explain why the construction in Figure 6 enforces the structure of a transitive tournament. Our construction produces encryptions of the signature and witness associated with each vertex. The crucial point is that transitions $\overline{m}_{j,i}$ are only allowed when $j < i$. Hence, $c_{j,i}^{(k)}$, $\mathrm{ct}_{j,i}$ and $\tilde{y}_{j,i}$ only exist if $j < i$. This guarantees that every vertex can be reached from any earlier vertex in the ordering, as well as there are no cycles.

Theorem 1. *For any $G := (V, E)$ satisfying Definition 1, the construction in Figure 6 satisfies graph enforcement (Definition 4) and verifiability (Definition 5) assuming that the adaptor signature scheme achieves full extractability and adaptability, the WES encryption scheme is IND-CPA and the discrete logarithm is hard, in the random oracle model.*

We provide some security intuitions below and defer the full details for the proof to Appendix B.

Graph enforcement (Definition 4). The adversary aims to extract a valid signature from c for vertex m_t by collecting attestations for subset of edges $E' \subset E$ that is not a path between m_0 and m_t. We consider the following attacks. First, the adversary might try to decrypt any of the ciphertexts $c_{j,t}^{(k)}$ without the corresponding attestation. Thus, the resulting $r_{j,t}^{(k)}$ can be used to derive the signature σ_t. Since WES achieves IND-CPA, this attack succeeds with negligible probability. Alternatively, the adversary could attempt to forge a signature using a presignature $\widetilde{\sigma}_{j,t}$. Since the adaptor signature scheme satisfies full extractability, this attack succeeds with negligible probability. Finally, the adversary might attempt to get the witnesses corresponding to some $(\tilde{Y}_{j,i}, Y_{j,i}, \mathsf{x}_i)$. Since the discrete logarithm problem is hard, this attack succeeds with negligible probability.

Verifiability (Definition 5). The adversary attempts to produce a malformed ciphertext that passes verification with VfEncGS, but causes the decryption to give an invalid signature. We consider the following attacks. The adversary encrypts 0 in all ciphertexts $c_{j,i}^{(k)}$ that go to the unopened set $\mathcal{S}_{\mathsf{uop}}$ and $r_{j,i}^{(k)}$ for those that are in the opened set $\mathcal{S}_{\mathsf{op}}$. Since $c_{j,i}^{(k)}$ is one of the inputs used to determine $(b_0, b_1, \ldots, b_\gamma)$, and H is modeled as a random oracle, the adversary would succeed with probability parametrized by γ. Concretely, the probability of succeeding is $2^{-\gamma}$, which is negligible for $\gamma = 128$ used in our implementation Section 4.3. Alternatively, the adversary produces an adaptor signature, such that the presignature verifies, but when adapted with the corresponding witness, it returns an invalid signature. Since the adaptor signature satisfies adaptability, this attack succeeds with negligible probability.

4.2 Construction for a Linear Graph

We now present a construction for a linear graph, which has a root vertex and every other node can be reached from it through exactly one path. An illustrative example of a linear graph is that composed by vertices m_0, m_1, m_2, m_3, and edges $\overline{m}_{0,1}, \overline{m}_{1,2}, \overline{m}_{2,3}$, as depicted in Figure 5. It is easy to see that the linear graph is a subset of the graphs supported by VGES according to Definition 1, since it is a directed acyclic graph and there is a path from m_0 to any other vertex.

As a naive approach, one could directly apply the construction in Figure 6 to encrypt signatures according to the logic of a linear graph. However, doing so would introduce a quadratic number of elements (in the number of vertices), since the construction iterates over all $j < i$ to account for every possible path leading to vertex m_i. In a linear graph, by contrast, each vertex is reachable through exactly one path. Therefore, the linear construction builds on Figure 6, but only has linear dependencies. We highlight next the main differences compared to Figure 6.

First, we require only a linear number of WES ciphertexts, as well as a linear number of y_i, $\tilde{y}_i$, ct_i, and $\widetilde{\sigma}_i$ elements. Hence, the construction does not require index j. Consequently, every $\tilde{y}_i$ is computed by sampling y_i and one-time padding it with w_0, $\tilde{y}_i := y_i + \mathsf{w}_0$. The opened and unopened sets are also reduced from quadratic size (in the number of vertices) to linear size. We compute each value $s_i^{(k)}$ as $s_i^{(k)} := r_1^{(k)} + \ldots + r_i^{(k)} + y_i$, which enables direct decryption using w_0 together with the attestations for all edges between m_0 and m_i. Because the graph is linear, decryption must process every WES ciphertext along this path, making the decryption cost linear in the number of vertices. We present the full details of the construction for the linear graph in Appendix C (Figure 12).

Theorem 2. *For any $G := (V, E)$ satisfying Definition 1, the construction in Appendix C satisfies graph enforcement (Definition 4) and verifiability (Definition 5) assuming that the adaptor signature scheme achieves full extractability and adaptability, the WES encryption scheme is IND-CPA and the discrete log problem is hard, in the random oracle model.*

Security holds for the same reasons as in the baseline construction. We defer the proof details to Appendix C.

4.3 Performance Evaluation

In this section, we evaluate the performance of the constructions presented in Sections 4.1 and 4.2. Our aim is to analyze their trade-offs in terms of required on-chain transactions, off-chain computation, and communication overhead for *bank* and *client* under

Public parameters: $(\mathbb{G}, g, q, \gamma, \mathsf{H}, N)$

$c \leftarrow \mathsf{EncGS}(sk, \overline{pk}, G := (V, E))$:

(1) Initialize $\mathcal{S}_{\mathsf{op}} := [\cdot]$; $\mathcal{S}_{\mathsf{uop}} := [\cdot]$ // Initialize opened and unopened sets
(2) Compute $\{(\mathsf{w}_i, \mathsf{x}_i)\}_{i=0}^{N} \leftarrow \mathsf{createR}^{N+1}(1^\lambda)$ // Prepare witnesses to enable graph navigation.
(3) Repeat for each $i \in [1, N]$: // Iterate over all vertices in the graph
  (a) Parse $m_i \in V$
  (b) Repeat for each $j \in [0, i\text{-}1]$ // Iterate over all edges directed to vertex i
    (i) Parse $\overline{m}_{j,i} \in E$
    (ii) Repeat for each $k \in [1, \gamma]$:
      – Sample $r_{j,i}^{(k)} \xleftarrow{\$} \mathbb{Z}_q$; and set $R_{j,i}^{(k)} := g^{r_{j,i}^{(k)}}$
      – Compute $c_{j,i}^{(k)} \leftarrow \mathsf{WES.Enc}((\overline{pk}, \overline{m}_{j,i}), r_{j,i}^{(k)}; \rho_{j,i}^{(k)})$; where $\rho_{j,i}^{(k)}$ are the random coins.
    (iii) Sample $y_{j,i} \xleftarrow{\$} \mathbb{Z}_q$, compute $\tilde{y}_{j,i} = \mathsf{w}_j + y_{j,i}$ and set $Y_{j,i} := g^{y_{j,i}}$ and $\tilde{Y}_{j,i} := g^{\tilde{y}_{j,i}}$
    (iv) Compute $\mathsf{ct}_{j,i} := \mathsf{w}_i + \tilde{y}_{j,i}$ and $\widetilde{\sigma}_{j,i} \leftarrow \mathsf{ADP.PreSig}(sk, m_i, \tilde{Y}_{j,i})$ // Encrypt signature & witness
    (v) Set $\Sigma_{j,i} := (\widetilde{\sigma}_{j,i}, \tilde{Y}_{j,i}, Y_{j,i})$
    (vi) Set $c_{j,i} := (\{c_{j,i}^{(k)}\}_{k=1}^{\gamma}, \mathsf{ct}_{j,i}, \{R_{j,i}^{(k)}\}_{k=1}^{\gamma}, \Sigma_{j,i})$
  (c) Set $c_i := \{c_{j,i}\}_{j=0}^{i-1}$
(4) Compute $(b_1, b_2, ..., b_\gamma) := \mathsf{H}(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0)$ // Determine indices to open
(5) Repeat for each $k \in [1, \gamma]$:
  – If $b_k = 1$, for all $i \in [1, N]$ and $j \in [0, i\text{-}1]$ $\mathcal{S}_{\mathsf{op}}[k, j, i] := (r_{j,i}^{(k)}, \rho_{j,i}^{(k)})$ // Reveal $r_{j,i}^{(k)}$ and $\rho_{j,i}^{(k)}$
  – If $b_k = 0$, for all $i \in [1, N]$ and $j \in [0, i\text{-}1]$ set $s_{j,i}^{(k)} := r_{j,i}^{(k)} + y_{j,i}$ and $\mathcal{S}_{\mathsf{uop}}[k, j, i] := s_{j,i}^{(k)}$ // Mask $y_{j,i}$
(6) Return $c := (\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0, \mathcal{S}_{\mathsf{op}}, \mathcal{S}_{\mathsf{uop}})$

$b := \mathsf{VfEncGS}(c, pk, \overline{pk}, G := (V, E))$:

(1) Parse $(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0, \mathcal{S}_{\mathsf{op}}, \mathcal{S}_{\mathsf{uop}}) \leftarrow c$
(2) Compute $(b_1, b_2, ..., b_\gamma) := \mathsf{H}(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0)$ // Determine indices to open
(3) Repeat for each $i \in [1, N]$: // Iterate over all vertices in the graph
  (a) Parse $\{c_{j,i}\}_{j=0}^{i-1} \leftarrow c_i$
  (b) Repeat for each $j \in [0, i\text{-}1]$ // Iterate over all edges directed to vertex i
    (i) Parse $m_i \in V$, $\overline{m}_{j,i} \in E$, $(\{c_{j,i}^{(k)}\}_{k=1}^{\gamma}, \mathsf{ct}_{j,i}, \{R_{j,i}^{(k)}\}_{k=1}^{\gamma}, \Sigma_{j,i}) \leftarrow c_{j,i}$, with $\Sigma_{j,i} := (\widetilde{\sigma}_{j,i}, \tilde{Y}_{j,i}, Y_{j,i})$
    (ii) Check that $g^{\mathsf{ct}_{j,i}} = \tilde{Y}_{j,i} \cdot \mathsf{x}_i$ and $\tilde{Y}_{j,i} = Y_{j,i} \cdot \mathsf{x}_j$ // Check that the ciphertext encrypts w_i
    (iii) Check that $\mathsf{ADP.PreVf}(pk, m_i, \tilde{Y}_{j,i}, \widetilde{\sigma}_{j,i}) = 1$. // Check that the presignature is valid
    (iv) Repeat for each $k \in [1, \gamma]$:
      – If $b_k = 1$, for $(r_{j,i}^{(k)}, \rho_{j,i}^{(k)}) \leftarrow \mathcal{S}_{\mathsf{op}}[k, j, i]$, check $c_{j,i}^{(k)} = \mathsf{WES.Enc}((\overline{pk}, \overline{m}_{j,i}), r_{j,i}^{(k)}; \rho_{j,i}^{(k)})$ // Check $c_{j,i}^{(k)}$
      – If $b_k = 0$, For $s_{j,i}^{(k)} \leftarrow \mathcal{S}_{\mathsf{uop}}[k, j, i]$, check $g^{s_{j,i}^{(k)}} = R_{j,i}^{(k)} \cdot Y_{j,i}$ // Check $s_{j,i}^{(k)}$
    (v) If any check fails abort and return 0
(4) Return 1

$\sigma_t := \mathsf{DecGS}(c, \{\overline{\sigma}_{j,i}\}_{E'})$:

(1) Parse $(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0, \mathcal{S}_{\mathsf{op}}, \mathcal{S}_{\mathsf{uop}}) \leftarrow c$.
(2) For each $\overline{m}_{j,i} \in E'$ and for each $k \in \mathcal{S}_{\mathsf{uop}}[k, j, i]$ do:
  (a) Parse $(\{c_{j,i}^{(k)}\}_{k=1}^{\gamma}, \mathsf{ct}_{j,i}, \{R_{j,i}^{(k)}\}_{k=1}^{\gamma}, \Sigma_{j,i}) \leftarrow c_{j,i}$, with $\Sigma_{j,i} := (\widetilde{\sigma}_{j,i}, \tilde{Y}_{j,i}, Y_{j,i})$
  (b) Compute $r_{j,i}^{(k)} := \mathsf{WES.Dec}(\overline{\sigma}_{j,i}, c_{j,i}^{(k)})$
  (c) If $g^{r_{j,i}^{(k)}} \neq R_{j,i}^{(k)}$, next k. // At least one $r_{j,i}^{(k)}$ in $\mathcal{S}_{\mathsf{uop}}$ is well-formed
  (d) Parse $s_{j,i}^{(k)} \leftarrow \mathcal{S}_{\mathsf{uop}}[k, j, i]$, compute $y_{j,i} := s_{j,i}^{(k)} - r_{j,i}^{(k)}$ and $\tilde{y}_{j,i} := y_{j,i} + \mathsf{w}_j$
  (e) If $\overline{m}_{j,i}$ not the last in E', compute $\mathsf{w}_i := \mathsf{ct}_{j,i} - \tilde{y}_{j,i}$, go to next $\overline{m}_{j,i} \in E'$
  (f) If $\overline{m}_{j,i}$ last in E' (i.e., $i = t$), compute $\sigma_t := \mathsf{ADP.Adapt}(\widetilde{\sigma}_{j,t}, \tilde{y}_{j,t})$
(3) Return σ_t

Figure 6: VGES **for a transitive tournament.**

different system parameters, as well as the overhead incurred by the oracle operation. While the off-chain computation and communication measurements relate directly to the constructions themselves, the evaluation of on-chain costs focuses on the repay, and withdraw stages described in Figure 4, which are the phases requiring to trigger events as encoded by VGES. All experiments were conducted on a Lenovo ThinkPad equipped with an Intel i7 processor running at 1.8 GHz and 16 GB of RAM.

Implementation. We have developed a proof-of-concept implementation in Rust[6] to evaluate the feasibility of our constructions. The implementation includes the constructions described in Figures 6 and 12, the main building block for collateralized loans. We have used the `bls12_381` and `secp256kfun` crates for pairing-based and elliptic curve-based cryptography, respectively, 1 and `bls12_381` crate for BLS signatures. We have implemented WES as in Figure 10 and Schnorr-based adaptor signatures as in [1]. Finally, oracle's attestations are realized using BLS signatures.

Implementation-level optimizations. We apply several optimizations to the encryption algorithm EncGS: (i) *precomputation*: Components c_1 and c_3, as well as randomness values r_1 and r_2 from the WES construction (cf. Figure 10), are precomputed before step 3.b.ii in Figure 6 and 3.b in Figure 12; (ii) *pairing reuse*: the pairing $e(\overline{pk}, H(\overline{m}_{j,i}))$ used to compute c_2 (cf. Figure 10) is computed once per $\overline{m}_{j,i} \in E$ in Figure 6 and once per $\overline{m}_{i-1,i} \in E$ in Figure 12, since it can be reused for all $k \in [1, \gamma]$; and (iii) *parallelization*: the computations at each iteration for $k \in [1, \gamma]$ run in parallel.

Number of on-chain transactions. We examine the number of on-chain transactions required in both $\mathcal{L}$ and $\widehat{\mathcal{L}}$ when using VGES. The results are summarized in Table 1. The two VGES constructions differ in the number of events that enable transitions between the SM states, which affects the total number of on-chain transactions required during the repayment stage. In the transitive tournament construction, the client may choose among multiple paths to reach the target terminal state, represented by transaction m_t. The shortest path involves satisfying only event $\overline{m}_{0,t}$, whereas the longest path requires satisfying the full sequence of events $\overline{m}_{0,1}$, $\overline{m}_{1,2}$, …, $\overline{m}_{t\text{-}1,t}$. In contrast, the linear construction provides only a single path to the target terminal state. Consequently, the client must satisfy all events in order ($\overline{m}_{0,1}$, $\overline{m}_{1,2}$, …, $\overline{m}_{t\text{-}1,t}$) regardless of circumstances. Finally, the withdraw stage requires only one transaction on $\mathcal{L}$, submitted either by the client (if at least one repayment has been made) or by the bank after the escrow times out.

Interestingly, the on-chain transactions in $\mathcal{L}$ can be further reduced if the client and the bank already have an operational payment channel (e.g., in the Lightning Network) between them in $\mathcal{L}$ with sufficient channel capacity. Hence, the withdraw transactions could be settled off-chain as a channel update in the payment channel, incurring in no on-chain costs. Transactions in ledger $\widehat{\mathcal{L}}$ cannot be settled in a payment channel, since the oracle must be able to read them on-chain to produce the corresponding attestations.

Off-chain computation (impact of N). The off-chain computation overhead for both bank and client is dominated by the encryption algorithm EncGS and the verification algorithm VfEncGS. These algorithms' performance is influenced by the number of vertices N in the repayment graph and the cut-and-choose security parameter γ. To analyze the impact of these parameters, we vary N from 3 to 24, representing short-term loans of 3 and 6 months and medium-term loans of 12 and 24 months, while fixing $\gamma = 128$ (64 bits of statistical soundness), and show in Figure 7 (left) the average over 10 repetitions. As expected in asymptotic terms, the computation overhead for the transitive tournament construction grows quadratically with N, while the linear construction growth is more linear. When checking concrete numbers, (e.g., at $N = 24$), the linear construction achieves up to a 10× speedup over the transitive tournament construction. The computation overhead of DecGS is significantly lower, so we have not plotted it in the graph, with average times ranging from 3.5ms for target node $N = 3$ to 6.5ms for $N = 24$.

[6] https://github.com/imdeadcm/collatloans

	R(TG)		R(LG)		W	
	$\mathcal{L}$	$\widehat{\mathcal{L}}$	$\mathcal{L}$	$\widehat{\mathcal{L}}$	$\mathcal{L}$	$\widehat{\mathcal{L}}$
Bank	0	0	0	0	1*	0
Client	0	$[1, t-1]$	0	$t-1$	1*	0

Table 1: Number of transactions required during the stages: (R)epayment (TG), (R)epayment (LG), and (W)ithdraw. TG denotes tournament graph and LG denotes linear graph. t is the index of the target terminal state. (W)ithdraw requires one transaction by the client or the bank if there was no repayment.

Off-chain computation (impact of γ). The cut-and-choose based proof in our constructions provides statistical soundness. Unlike computational soundness, statistical soundness ensures that even an adversary with unbounded computational power gains no advantage beyond the stated probability of failure to break the soundness of the proof system. Consequently, the parameter γ can be tuned to match the desired security level of cut-and-choose. To analyze the impact of parameter γ on off-chain computation, we fix $N = 24$ and vary γ among three values: $\gamma = 80$, $\gamma = 128$, and $\gamma = 256$, corresponding to 40, 64, and 128 bits of statistical soundness, respectively. We repeated the experiment 10 times and computed that average values. The results are reported in Figure 7 (center). We observe that setting $\gamma = 128$ yields a 50% reduction in computation time compared to $\gamma = 256$, while still achieving 64 bits of statistical soundness, which is sufficient for practical deployments.

Communication complexity. The communication cost, shown in Figure 7 (right) for $\gamma = 128$, is dominated by the exchange of the VGES ciphertext, which in our constructions corresponds to the ciphertext c produced by EncGS. Analytically, in both constructions the communication overhead is $|\{c_i\}_{i=1}^{N}| + |\{\mathsf{x}_i\}_{i=0}^{N}| + |\mathsf{w}_0| + |\mathcal{S}_{\mathsf{op}}| + |\mathcal{S}_{\mathsf{uop}}|$. Components $|\{\mathsf{x}_i\}_{i=0}^{N}|$ and $|\mathsf{w}_0|$, have identical size in both constructions. However, $|\mathcal{S}_{\mathsf{op}}|$ and $|\mathcal{S}_{\mathsf{uop}}|$ grow quadratically in N for transitive tournament graphs, but only linearly in for linear graphs. Similarly, the dominant contribution to $|\{c_i\}_{i=1}^{N}|$ comes from the WES ciphertexts, which exhibit quadratic growth in N in the construction for tournament graphs and linear growth in the construction for linear graphs. Overall, for $N = 24$ and $\gamma = 128$, communication cost remains bounded by 30 megabytes when considering a transitive tournament graph and 3 megabytes for the case of linear graphs. Recall that this is a one time message (i.e., c) exchanged at the beginning of the protocol (c.f. Figure 4), similar to how currently a client would download the terms and conditions of the loan from the bank's website before signing the contract.

Overhead for the oracle functionality. The oracle functionality requires signing every transaction in a block. We evaluate the overhead of this functionality by measuring the computational and storage costs of signing all transactions in a block. On average, a Bitcoin block contains 3000 transactions [5]. In our evaluation, the

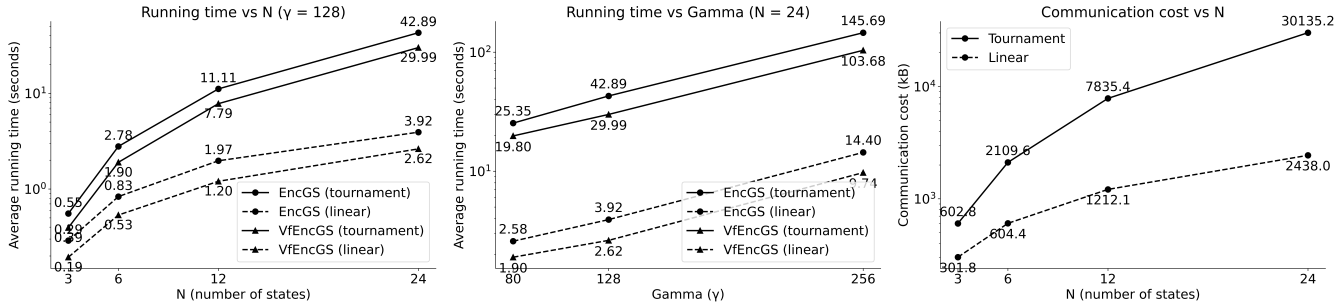


Figure 7: Evaluation of VGES**. Left: running time of algorithms** EncGS **and** VfEncGS **vs number of repayment states (N). Center: running time of algorithms** EncGS **and** VfEncGS **vs parameter γ of cut-and-choose. Right: communication cost in kB.**

oracle incurs a computational cost of 2.6 seconds per block, with a total storage cost of 288kB to store all attestations for a block. Consequently, storing attestations for transactions generated over the preceding month, corresponding to approximately 4500 blocks, requires 1265MB.

Summary. Our evaluation shows that both constructions are practical, with clear trade-offs. The construction for a transitive tournament allows flexible repayment paths but incurs quadratic off-chain computation and communication costs in the length of the graph. In contrast, the construction for a linear graph fixes the number of on-chain repayment transactions while reducing the dominant off-chain costs from quadratic to linear in N, achieving up to a 10× speedup and significantly lower communication overhead. The cut-and-choose parameter γ enables a tunable security–performance trade-off, with $\gamma = 128$ offering an effective balance. Finally, the overhead of the oracle functionality is modest, requiring only seconds of computation per block and storage on the order of hundreds of megabytes per month, demonstrating feasibility.

5 Discussion

In this section, we discuss practical considerations relevant to the deployment of our protocol. In Appendix D we discuss how to extend our protocol to support dynamic exchange rate between blockchains $\mathcal{L}$ and $\widehat{\mathcal{L}}$, as well as privacy considerations.

5.1 Why an Oracle and not a Trusted Party?

Our construction builds upon that transactions occurring in $\widehat{\mathcal{L}}$ are attested so that the client can use these attestations to decrypt a ciphertext. Thus, system security relies on the oracle not colluding with either the bank or the client. A natural question is why, given this assumption, we do not simply trust the oracle fully and use it as the trusted intermediary for the loan. The first reason is scalability: an oracle that only produces attestations for transactions included in the blockchain does not need to scale its computational resources with the number of clients and banks running loans in parallel. Moreover, the same oracle can be reused across many decentralized finance applications that also require an oracle attesting to the inclusion of transactions in the blockchain (e.g., MixBuy [9]). By contrast, a trusted intermediary would need to scale its computational infrastructure with the number of clients and banks it serves. The second reason is the ease of decentralization. As we discuss next in Section 5.2, the oracle functionality can be readily decentralized using threshold techniques, either on the oracle side or within the VGES scheme itself. This allows the oracle to be distributed across many parties, making it significantly harder for a malicious coalition to reach the threshold. By contrast, decentralizing a trusted party that executes a complex application requires running a multi-party computation (MPC) protocol, which is typically limited in scale; practical deployments of MPC rarely exceed three parties. Finally, there is the cost of accountability. For an oracle attesting to transactions, identifying misbehavior is possible, as we discuss in more detail in Section 5.3, since blockchain $\widehat{\mathcal{L}}$ maintains a public record of the transactions the oracle is expected to have attested. By contrast, determining whether a trusted party executing the contract logic has misbehaved may require complex legal analysis, whose complexity grows with the sophistication of the application it operates.

5.2 Distributing the Oracle Functionality

For ease of exposition, we presented our solution as if the oracle was a single party trusted by both bank and client. However, practical deployments would decentralize this trust across multiple oracles. We discuss two approaches.

Threshold-signatures. A set of oracles jointly run a threshold signature scheme, such as threshold BLS [14], with threshold τ. Each oracle publishes its signature share on the attestation message, and users collect any τ shares to reconstruct an attestation. This approach introduces no additional computational or communication overhead for bank or client beyond what we report in Section 4.3. This approach however, sets the oracles and the threshold as a system parameter, and addition or removal of oracles requires a redistribution of keys for all oracles.

Threshold-encryption. Alternatively, the oracles can operate fully independently, each producing its own attestation. At setup time, bank and client choose the set of oracles they trust and specify the minimum number τ of oracle signatures required for a transaction in $\widehat{\mathcal{L}}$ to be considered *attested*. The protocol then adapts as follows (described for Figure 6, with an analogous change in Figure 12): in step 3.b.ii, instead of directly encrypting r_i, the bank secret-shares r_i using a τ-out-of-n sharing scheme, then encrypts each share under the public key of a different oracle through the WES encryption. Following the performance evaluation in [31], which

uses the threshold encryption that we described above, we estimate that with seven oracles and threshold $\tau = 4$, the computational overhead increases by approximately a factor of two which is still practical on commodity hardware.

5.3 Oracle Accountability

Consider that a bank and client setup a loan using VGES following Figure 4. Both the bank and the client can present conclusive evidence to identify misbehaving oracles.

Bank blaming the oracle. Consider that the client has not repaid any repayment transaction in ledger $\widehat{\mathcal{L}}$. If the bank detects that transaction m_t has been accepted in ledger $\mathcal{L}$, allowing the client to claim back any collateral, this indicates misbehavior by the oracle. Given that VGES satisfies graph enforcement the client could only have obtained the signature corresponding to m_t if the oracle gave them the corresponding attestation(s) leading from the root node m_0 to the target node m_t. Therefore, the bank can use the transaction history in $\mathcal{L}$ and $\widehat{\mathcal{L}}$ to prove misbehavior by the oracle.

Client blaming the oracle. Conversely, suppose the client submits to $\widehat{\mathcal{L}}$ the transaction corresponding to the transition $\overline{m}_{0,t}$, but does not get the signature corresponding to m_t. Since VGES satisfies verifiability, the client could only not get the signature if the oracle did not provide the attestation corresponding to $\overline{m}_{0,t}$. In this case, the client can use the transactions recorded on $\widehat{\mathcal{L}}$ to demonstrate misbehavior by the oracle.

In both scenarios, the bank and the client can use the transaction history of $\mathcal{L}$ and $\widehat{\mathcal{L}}$ as verifiable evidence of misbehavior by the oracle. Such evidence can serve as the basis for a slashing mechanism to penalize faulty oracles. Designing such a mechanism is an interesting direction for future work, particularly since the oracle functionality is also employed in other recent works [9, 15, 31].

5.4 Using VGES in Other Applications

The functionality provided by VGES can be leveraged in applications other than collateralized loans as follows: (i) encode the SM as a graph $G := (V, E)$; (ii) use VGES to encrypt signatures according to G; (iii) collect attestations from the oracle on a subset of edges $E' \subset E$; (iv) if E' is a path from m_0 to m_t, decrypt the signature on m_t. See examples below.

Contractual payment obligation. A client and a contractor working on a multi-milestone project can establish an escrow account in which the client locks the full project budget [41]. The vertices V represent the fractions of the budget that the contractor can claim after completing each milestone in the project (i.e. the edges E in the graph). The oracle attests to milestone completion. The contractor claims budget proportional to the attestations collected.

Asset vesting. To retain talent, an employer may lock contributions to an employee's pension plan in an escrow account, which become claimable only after the employee completes a minimum period at the company [28] (e.g., two years). The vertices V represent incremental fractions of the employer's contributions that the employee can claim, while the edges E correspond to time intervals that unlock additional contributions. The oracle attests, at regular intervals, the accumulated employment time for each employee. The employee can claim contributions only after collecting attestations covering the required period.

Rental agreement. In a rental contract, a deposit is locked until the renter returns the property. For example, consider a car rental [17, 42]. The vertices V represent incremental fractions of the deposit the renter can claim after returning the vehicle. The edges E correspond to conditions required to recover the deposit, such as returning a full fuel tank or ensuring no damage to the car. The oracle attests that these conditions have been met. The renter can reclaim the full deposit only after collecting attestations for all conditions.

6 Comparison with Previous Work

Verifiable witness encryption. Döttling et al. [15] and Madathil et al. [31] introduced verifiable witness encryption schemes based on digital signatures. While these prior works achieve verifiable witness encryption, they fall short in our setting. We encode our applications as a graph $G = (V, E)$ and verifiably encrypt signatures for messages corresponding to each vertex $m_i \in V$. A ciphertext can be decrypted *iff* the decryptor has collected attestations on a subset of edges $E' \subset E$ that form a path from m_0 to m_t. Achieving this dependency using the approaches in [15, 31] would be cryptographically inefficient. A naïve adaptation would require a complex witness encryption scheme in which the decryptor must produce a witness to a language asserting that the ciphertext corresponding to a previous vertex was successfully decrypted, an approach that is both inefficient and impractical. Our protocol extends prior solutions by embedding, alongside each signature, a second witness that is unlocked only when a preceding ciphertext is opened. Concretely, decrypting a ciphertext c_i corresponding to vertex $m_i \in V$ requires (i) an attestation from the oracle on an edge $\overline{m}_{j,i} \in E$; and (ii) a witness to a discrete-log relation, made available when decrypting c_j, corresponding to $m_j \in V$. In effect, we extend the constructions of [15, 31] so that ciphertexts are decryptable only according to the logic of the SM, which necessitates new security definitions capturing these additional dependencies.

Glimpse. Glimpse [40] proposes an atomic swap protocol for Bitcoin based on Merkle proofs executed through Bitcoin scripts (i.e., Bitcoin-based smart contracts). One of the use cases of their protocol is to enable loans in Bitcoin. However, their approach allows the client to repay the loan in a single transaction, thus not supporting repayments following the logic of the SM discussed in our work.

7 Conclusions and Future Work

We show that collateralized loans can be implemented on blockchains securely and efficiently without relying on smart contracts. Our novel primitive verifiable graph encryption for signatures (VGES) enables the release of collateral according to a graph that captures the logic of the SM of the loan contract. We formally prove the security of VGES for our constructions and demonstrate its efficiency through a Rust-based implementation. Beyond loans, VGES also supports other applications, such as milestone-based payments, asset vesting, and rental agreements. As future work, we plan to extend our primitive to support a broader class of graphs beyond Definition 1, making VGES compatible with more applications.

Acknowledgments

This work is supported by projects PRODIGY (TED2021-132464B-I00) and ESPADA (PID2022-142290OB-I00), which are funded by

MCIN/AEI/10.13039/501100011033/ and the European Union NextGenerationEU / PRTR, and by the Excellence Seal María de Maeztu with grant number CEX2024-001471-M, which is funded by MICIU/AEI/10.13039/501100011033, by the Luxembourg National Research Fund (FNR), project CryptoFin C22/IS/17415825, by the FutureFinTech National Centre of Excellence in Research and Innovation, FNR grant reference NCER22/IS/16570468/NCER-FT. Sri AravindaKrishnan Thyagarajan was supported by Sui Foundation Academic Research Program 2025.

References

[1] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. 2021. Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures. In *ASIACRYPT 2021, Part II (LNCS, Vol. 13091)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Cham, 635–664. https://doi.org/10.1007/978-3-030-92075-3_22

[2] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. 2021. Blitz: Secure Multi-Hop Payments Without Two-Phase Commits. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 4043–4060. https://www.usenix.org/conference/usenixsecurity21/presentation/aumayr

[3] Waclaw Banasik, Stefan Dziembowski, and Daniel Malinowski. 2016. Efficient Zero-Knowledge Contingent Payments in Cryptocurrencies Without Scripts. In *ESORICS 2016, Part II (LNCS, Vol. 9879)*, Ioannis G. Askoxylakis, Sotiris Ioannidis, Sokratis K. Katsikas, and Catherine A. Meadows (Eds.). Springer, Cham, 261–280. https://doi.org/10.1007/978-3-319-45741-3_14

[4] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 459–474. https://doi.org/10.1109/SP.2014.36

[5] Blockchain.com. [n. d.]. Number of Transactions per Block. https://www.blockchain.com/explorer/charts/n-transactions-per-block. Accessed: 2026-04-29.

[6] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2020. Zether: Towards Privacy in a Smart Contract World. In *FC 2020 (LNCS, Vol. 12059)*, Joseph Bonneau and Nadia Heninger (Eds.). Springer, Cham, 423–443. https://doi.org/10.1007/978-3-030-51280-4_23

[7] Sergiu Bursuc and Sjouke Mauw. 2022. Contingent payments from two-party signing and verification for abelian groups. In *CSF 2022 Computer Security Foundations Symposium*. IEEE Computer Society Press, 195–210. https://doi.org/10.1109/CSF54842.2022.9919674

[8] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-Knowledge Contingent Payments Revisited: Attacks and Payments for Services. In *ACM CCS 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM Press, 229–243. https://doi.org/10.1145/3133956.3134060

[9] Diego Castejon-Molina, Dimitrios Vasilopoulos, and Pedro Moreno-Sanchez. 2025. MixBuy: Contingent Payment in the Presence of Coin Mixers. *PoPETs* 2025, 1 (Jan. 2025), 671–706. https://doi.org/10.56553/popets-2025-0036

[10] CoinMarketCap. 2026. Cryptocurrency Market Capitalizations. https://coinmarketcap.com/. Accessed: 2026-04-27.

[11] Wei Dai, Tatsuaki Okamoto, and Go Yamamoto. 2022. Stronger Security and Generic Constructions for Adaptor Signatures. In *INDOCRYPT 2022 (LNCS, Vol. 13774)*, Takanori Isobe and Santanu Sarkar (Eds.). Springer, Cham, 52–77. https://doi.org/10.1007/978-3-031-22912-1_3

[12] Aave DAO. 2024. Aave V3 Origin Repository. https://github.com/aave-dao/aave-v3-origin/tree/main. Accessed: 2025-03-19.

[13] Aave DAO. 2024. Aave V3 Technical Paper. https://github.com/aave-dao/aave-v3-origin/blob/main/docs/Aave_V3_Technical_Paper.pdf. Accessed: 2025-03-19.

[14] Sourav Das and Ling Ren. 2024. Adaptively Secure BLS Threshold Signatures from DDH and co-CDH. In *CRYPTO 2024, Part VII (LNCS, Vol. 14926)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 251–284. https://doi.org/10.1007/978-3-031-68394-7_9

[15] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. 2023. McFly: Verifiable Encryption to the Future Made Practical. In *FC 2023, Part I (LNCS, Vol. 13950)*, Foteini Baldimtsi and Christian Cachin (Eds.). Springer, Cham, 252–269. https://doi.org/10.1007/978-3-031-47754-6_15

[16] European Central Bank. 2026. ECB Paves Way for Acceptance of DLT-Based Assets as Eligible Eurosystem Collateral. https://www.ecb.europa.eu/press/pr/date/2026/html/ecb.pr260127_1~a946167ce1.en.html. Press release, European Central Bank.

[17] European Consumer Centres Network. [n. d.]. Car Rental Rights. https://www.eccnet.eu/consumer-rights/what-are-my-consumer-rights/travel-and-passenger-rights/road-travel-rights/car. Accessed: 2026-04-29.

[18] Compound Finance. 2024. Compound Comet Repository. https://github.com/compound-finance/comet. Accessed: 2025-03-19.

[19] Compound Finance. 2024. Compound Finance Documentation - Protocol Contracts. https://docs.compound.finance/#protocol-contracts. Accessed: 2025-03-19.

[20] Georg Fuchsbauer. 2019. WI Is Not Enough: Zero-Knowledge Contingent (Service) Payments Revisited. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 49–62. https://doi.org/10.1145/3319535.3354234

[21] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. 2013. Witness encryption and its applications. In *45th ACM STOC*, Dan Boneh, Tim Roughgarden, and Joan Feigenbaum (Eds.). ACM Press, 467–476. https://doi.org/10.1145/2488608.2488667

[22] Paul Gerhart, Dominique Schröder, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. 2024. Foundations of Adaptor Signatures. In *EUROCRYPT 2024, Part II (LNCS, Vol. 14652)*, Marc Joye and Gregor Leander (Eds.). Springer, Cham, 161–189. https://doi.org/10.1007/978-3-031-58723-8_6

[23] Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. 2022. Foundations of Coin Mixing Services. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 1259–1273. https://doi.org/10.1145/3548606.3560637

[24] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks. *SIAM J. Comput.* 17, 2 (April 1988), 281–308.

[25] Javier Gomez-Martinez, Dimitrios Vasilopoulos, Pedro Moreno-Sanchez, and Dario Fiore. 2025. Algebraic Zero Knowledge Contingent Payment. In *ACNS 2025, Part II (LNCS, Vol. 15826)*, Marc Fischlin and Veelasha Moonsamy (Eds.). Springer, Cham, 224–254. https://doi.org/10.1007/978-3-031-95764-2_10

[26] David Han and Coinbase Institutional Research. 2025. The Lending Landscape. https://www.coinbase.com/es-es/institutional/research-insights/research/market-intelligence/the-lending-landscape. Accessed: 2025-10-09.

[27] Lucjan Hanzlik, Julian Loss, Sri Aravinda Krishnan Thyagarajan, and Benedikt Wagner. 2024. Sweep-UC: Swapping Coins Privately. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 3822–3839. https://doi.org/10.1109/SP54263.2024.00081

[28] Investopedia. 2023. Vesting: What It Is and How It Works. https://www.investopedia.com/terms/v/vesting.asp. Accessed: 2025-03-31.

[29] Liquity. 2024. Liquity BOLD - Borrower Operations. https://github.com/liquity/bold/?tab=readme-ov-file#borroweroperations. Accessed: 2025-03-19.

[30] Liquity. 2024. Liquity V2 FAQ - General. https://docs.liquity.org/v2-faq/general. Accessed: 2025-03-19.

[31] Varun Madathil, Sri Aravinda Krishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. 2023. Cryptographic Oracle-based Conditional Payments. In *NDSS 2023*. The Internet Society.

[32] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. 2019. Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In *NDSS 2019*. The Internet Society. https://doi.org/10.14722/ndss.2019.23330

[33] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2016. A fistful of Bitcoins: characterizing payments among men with no names. *Commun. ACM* 59, 4 (March 2016), 86–93. https://doi.org/10.1145/2896384

[34] Morpho Labs. 2025. Morpho Blue Protocol. https://github.com/morpho-org/morpho-blue/tree/731e3f7ed97cf15f8fe00b86e4be5365eb3802ac. Accessed: 2025-10-23.

[35] Ky Nguyen, Miguel Ambrona, and Masayuki Abe. 2020. WI is Almost Enough: Contingent Payment All Over Again. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 641–656. https://doi.org/10.1145/3372297.3417888

[36] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. 2019. Confidential Assets. In *FC 2018 Workshops (LNCS, Vol. 10958)*, Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore, and Massimiliano Sala (Eds.). Springer, Berlin, Heidelberg, 43–63. https://doi.org/10.1007/978-3-662-58820-8_4

[37] Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oguzhan Ersoy, Amin Sakzad, Muhammed F. Esgin, Joseph K. Liu, Jiangshan Yu, and Tsz Hon Yuen. 2023. BlindHub: Bitcoin-Compatible Privacy-Preserving Payment Channel Hubs Supporting Variable Amounts. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2462–2480. https://doi.org/10.1109/SP46215.2023.10179427

[38] Fergal Reid and Martin Harrigan. 2011. An Analysis of Anonymity in the Bitcoin System. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. 1318–1326. https://doi.org/10.1109/PASSAT/SocialCom.2011.79

[39] Dorit Ron and Adi Shamir. 2013. Quantitative Analysis of the Full Bitcoin Transaction Graph. In *FC 2013 (LNCS, Vol. 7859)*, Ahmad-Reza Sadeghi (Ed.). Springer, Berlin, Heidelberg, 6–24. https://doi.org/10.1007/978-3-642-39884-1_2

[40] Giulia Scaffino, Lukas Aumayr, Zeta Avarikioti, and Matteo Maffei. 2023. Glimpse: On-Demand PoW Light Client with Constant-Size Storage for DeFi. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 733–750. https://www.usenix.org/conference/usenixsecurity23/presentation/scaffino
[41] Sirion. [n. d.]. Contract Payment. https://www.sirion.ai/library/contract-negotiation/contract-payment/. Accessed: 2026-04-29.
[42] SIXT. [n. d.]. Deposit Refund. https://www.sixt.com/help-center/articles/deposit-refund/. Accessed: 2026-04-29.
[43] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. 2021. A^{2}L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1834–1851. https://doi.org/10.1109/SP40001.2021.00111
[44] Ertem Nusret Tas, István András Seres, Yinuo Zhang, Márk Melczer, Mahimna Kelkar, Joseph Bonneau, and Valeria Nikolaenko. 2024. Atomic and Fair Data Exchange via Blockchain. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 3227–3241. https://doi.org/10.1145/3658644.3690248
[45] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. 2022. Universal Atomic Swaps: Secure Exchange of Coins Across All Blockchains. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1299–1316. https://doi.org/10.1109/SP46214.2022.9833731
[46] Florian Tramèr, Dan Boneh, and Kenny Paterson. 2020. Remote Side-Channel Attacks on Anonymous Transactions. In *USENIX Security 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2739–2756. https://www.usenix.org/conference/usenixsecurity20/presentation/tramer

A Extended preliminaries

Digital signature scheme. We require a digital signature scheme DS:=(KGen, Sig, Vf), where: (i) KGen gets as input 1^λ and outputs a key pair (pk, sk); (ii) Sig gets as input sk and a message m, and outputs a signature σ; and (iii) Vf is deterministic, gets as input pk, the message m, and the signature σ, and outputs a bit b. We assume that DS is correct (i.e., it holds that $\mathsf{Vf}(pk, m, \mathsf{Sig}(sk, m)) = 1$ if σ is computed as above) and secure under the standard notion of existential unforgeability under chosen message attack (EUF-CMA) [24], which we restate in Figure 8.

Definition 6 (EUF-CMA)**.** *A digital signature scheme offers EUF-CMA if for all $\lambda \in \mathbb{N}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all PPT adversaries $\mathcal{A}$, it holds that* $\Pr[\mathsf{ExpForge}(\lambda) = 1] \leq \mathsf{negl}$*, with* ExpForge *in Figure 8.*

$\mathsf{ExpForge}(\lambda)$	$\mathcal{O}\mathsf{Sig}(m)$
$Q := \emptyset$	$\sigma \leftarrow \mathsf{Sig}(sk, m)$
$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$	$Q := Q \cup m$
$(m, \sigma) \leftarrow \mathcal{A}^{\mathcal{O}\mathsf{Sig}}(pk)$	**return** σ
return $\mathsf{Vf}(pk, m, \sigma) \wedge m \notin Q$	

Figure 8: Experiment for signature unforgeability .

Bilinear pairing. Let $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ be three prime order cyclic groups with scalar field $\mathbb{Z}_q$, and let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be generators. A pairing is an efficiently computable function $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ satisfying: (i) *bilinearity:* For all $u, u' \in \mathbb{G}_1$ and $v, v' \in \mathbb{G}_2$ we have $e(u \cdot u', v) = e(u, v) \cdot e(u', v)$ and $e(u, v \cdot v') = e(u, v) \cdot e(u, v')$; (ii) *non-degeneracy:* $g_T = e(g_1, g_2)$ is a generator of $\mathbb{G}_T$; and (iii) *computability:* For all $u \in \mathbb{G}_1$ and $v \in \mathbb{G}_2$, $e(u, v)$ is efficiently computable.

Witness encryption based on signatures. We require a witness encryption based on signatures scheme WES := (Enc, Dec), defined with respect to a digital signature scheme $\overline{\mathsf{DS}} = (\overline{\mathsf{KGen}}, \overline{\mathsf{Sig}}, \overline{\mathsf{Vf}})$, where: (i) PPT algorithm Enc gets as input a tuple comprising a verification key $\overline{pk}$ and a message $\overline{m}$, a plaintext m, and outputs a ciphertext c; and (ii) DPT algorithm Dec gets as input a signature $\hat{\sigma}$ and a ciphertext c, and outputs a plaintext m. We say that WES is correct if it holds that $\Pr[\mathsf{Dec}(\overline{\mathsf{Sig}}(\overline{sk}, \overline{m}), \mathsf{Enc}((\overline{pk}, \overline{m}), m)) = m] = 1$, and we require the security notion of indistinguishability under chosen plaintext attack (IND-CPA) as defined in [31]. We now restate WES IND-CPA.

Definition 7 (IND-CPA)**.** WES *offers IND-CPA if for all $\lambda \in \mathbb{N}$, there exists a negligible function* $\mathsf{negl}(\lambda)$ *such that for all PPT adversaries $\mathcal{A}$, it holds that* $\Pr[\mathsf{IND\text{-}CPA}(\lambda) = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$*, with* IND-CPA *in Figure 9.*

$\mathsf{IND\text{-}CPA}(\lambda)$	$\mathcal{O}\overline{\mathsf{Sig}}(\overline{m})$
$Q := \emptyset$; $b \xleftarrow{\$} \{0,1\}$	$\overline{\sigma} \leftarrow \overline{\mathsf{Sig}}(\overline{sk}, \overline{m})$
$(\overline{pk}, \overline{sk}) \leftarrow \overline{\mathsf{KGen}}(1^\lambda)$	$Q := Q \cup \{\overline{m}\}$
$(\overline{m}^*, m_0, m_1) \leftarrow \mathcal{A}^{\mathcal{O}\overline{\mathsf{Sig}}}(\overline{pk})$	**return** $\overline{\sigma}$
$c_b \leftarrow \mathsf{Enc}((\overline{pk}, \overline{m}^*), m_b)$	
$b' \leftarrow \mathcal{A}^{\mathcal{O}\overline{\mathsf{Sig}}}(c_b)$	
return $(b = b') \wedge (\overline{m}^* \notin Q)$	

Figure 9: Experiment IND-CPA for witness encryption based on signatures.

We restate in Figure 10 the WES encryption construction used in [31], used in our implementation.

Adaptor signature scheme. We require an adaptor signature scheme [1, 11, 22] ADP:=(PreSig, PreVf, Adapt, Extract), defined with respect to a digital signature scheme DS and a relation $\mathcal{R}$ where: (i) PPT algorithm PreSig gets as input a signing key sk, a message m, and a public statement x, and outputs a pre-signature $\widetilde{\sigma}$; (ii) DPT algorithm PreVf gets as input a verification pk, a message m, a public statement x and a pre-signature $\widetilde{\sigma}$ and outputs 1 if $\widetilde{\sigma}$ is a valid pre-signature on m under pk and x, otherwise it outputs 0; (iii) DPT algorithm Adapt gets as input a pre-signature $\widetilde{\sigma}$ and a witness w, and outputs a signature σ; and (iv) DPT algorithm Extract gets as input a signature σ, a pre-signature $\widetilde{\sigma}$ and a public statement x, and outputs a witness w. We require ADP to be correct and secure for full extractability, adaptability and pre-verify soundness, as defined in [11, 22].

We say an adaptor signature scheme is *correct* if for all $\lambda \in \mathbb{N}$, all m, the following holds:

$$\Pr\left[\begin{array}{r|c} & (\mathsf{x}, \mathsf{w}) \leftarrow \mathsf{createR}(1^\lambda) \\ & (pk, sk) \leftarrow \mathsf{KGen}(1^\lambda) \\ b_0 = 1 & \widetilde{\sigma} \leftarrow \mathsf{PreSig}(sk, m, \mathsf{x}) \\ \wedge\ b_1 = 1 & \sigma \leftarrow \mathsf{Adapt}(\widetilde{\sigma}, \mathsf{w}) \\ \wedge\ b_2 = 1 & \mathsf{w}^* \leftarrow \mathsf{Extract}(\sigma, \widetilde{\sigma}, \mathsf{x}) \\ & b_0 := \mathsf{PreVf}(pk, m, \mathsf{x}, \widetilde{\sigma}) \\ & b_1 := \mathsf{Vf}(pk, m, \sigma);\ b_2 := (\mathsf{x}, \mathsf{w}^*) \in \mathcal{R} \end{array}\right] = 1$$

Definition 8 (Full Extractability)**.** *An adaptor signature scheme is said to offer full extractability if for all $\lambda \in \mathbb{N}$, there exists a negligible*

$\underline{\mathsf{WES.Enc}((\overline{pk}, \overline{m}), m)}$:
- Sample $r_1 \leftarrow \mathbb{Z}_q, r_2 \leftarrow \mathbb{G}_T$
- Set $c_1 := g_1^{r_1}, h := H(r_2)$
- Set $c_2 := (e(\overline{pk}, H_2(\overline{m}))^{r_1} \cdot r_2)$
- Set $c_3 = h + m$
- Return (c_1, c_2, c_3)

$\underline{\mathsf{WES.Dec}(\overline{\sigma}, (c_1, c_2, c_3))}$:
- Compute $r = c_2 \cdot e(c_1, \sigma)^{-1}$
- Compute $h = H(r)$
- Return $m = c_3 - m$

Figure 10: Witness encryption based on BLS signatures [31], where H, H_2 are hash functions defined as $H : \mathbb{G}_T \rightarrow \{0,1\}^\lambda$; $H_2 : \{0,1\}^\lambda \rightarrow \mathbb{G}_2$.

function $\mathsf{negl}(\lambda)$ *such that for all PPT adversaries* $\mathcal{A}$*, it holds that* $\Pr[(\mathsf{s})\mathsf{fext}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$*, where* $(\mathsf{s})\mathsf{fext}$ *is defined in Figure 11.*

$\underline{\mathsf{fext}(\lambda)}$
$Q_{fext} := \emptyset$; $Q_{st} := \emptyset$; $Q_{pS} := [\,]$
$(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$
$(m^*, \sigma^*) \leftarrow \mathcal{A}^{O\mathsf{Sig}, O\mathsf{PreSig}, O\mathsf{newx}}(pk)$
assert $m* \notin Q_{fext}$
$b_0 := \mathsf{Vf}(pk, m^*, \sigma^*)$
$b_1 := \forall(\mathsf{x}, \widetilde{\sigma}) \in Q_{pS}[m^*]\ s.t.\ \mathsf{x} \notin Q_{st}$
$(\mathsf{x}, \mathsf{Extract}(\sigma^*, \widetilde{\sigma}, \mathsf{x})) \notin \mathcal{R}$
return $b_0 \wedge (b_1 \vee b_2)$

$\underline{O\mathsf{Sig}(m)}$
$\sigma \leftarrow \mathsf{Sig}(sk, m)$
$Q_{fext} := Q_{fext} \cup \{m\}$
return σ

$\underline{O\mathsf{PreSig}(m, \mathsf{x})}$
$\widetilde{\sigma} \leftarrow \mathsf{PreSig}(sk, m, \mathsf{x})$
$Q_{pS}[m] := Q_{pS}[m] \cup \{(\mathsf{x}, \widetilde{\sigma})\}$
return $\widetilde{\sigma}$

$\underline{O\mathsf{newx}()}$
$(\mathsf{x}, \mathsf{w}) \leftarrow \mathsf{createR}(1^\lambda)$
$Q_{st} := Q_{st} \cup \{\mathsf{x}\}$
return x

Figure 11: Experiment for full extractability.

Definition 9 (Pre-Signature Adaptability)**.** *An adaptor signature scheme offers pre-signature adaptability if for all* $\lambda \in \mathbb{N}$*, any message* $m \in \{0,1\}^*$*, any statement and witness pair* $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$*, any public key such that* $pk \in \mathsf{SUPP}(\mathsf{KGen})$ *and any pre-signature* $\widetilde{\sigma} \in \{0,1\}^*$ *that satisfies* $\mathsf{PreVf}(pk, m, \mathsf{x}, \widetilde{\sigma})$*, we have that*
$\Pr[\mathsf{Vf}(pk, m, \mathsf{Adapt}(\widetilde{\sigma}, \mathsf{w})) = 1] = 1.$

Definition 10 (Pre-Verify Soundness)**.** *An adaptor signature scheme satisfies pre-verify soundness if for every PPT adversary* $\mathcal{A}$ *there exists a negligible function* negl *such that for every* $\lambda \in \mathbb{N}$ *and polynomially bounded* $\mathsf{x} \notin \mathcal{R}$*,*

$$\Pr\left[\ \mathsf{PreVf}(pk, m, \mathsf{x}, \widetilde{\sigma}) = 1 \ \middle|\ \begin{matrix}(pk, sk) \leftarrow \mathsf{KGen}(1^n) \\ (m, \widetilde{\sigma}) \leftarrow \mathcal{A}(sk)\end{matrix}\right] \leq \mathsf{negl}(\lambda).$$

B Proof Of Theorem 1

Proposition B.1. *Assume that the adaptor signature scheme achieves full extractability, the WES encryption scheme is IND-CPA and the discrete logarithm is hard in the random oracle model. Then, the transitive tournament construction satisfies graph enforcement.*

Proof. Towards a contradiction, we present a series of hybrids until we reduce the last one to hardness of the discrete logarithm problem.
Hybrid$_0$: This is the experiment defined in Definition 4.
Hybrid$_1$: This hybrid is the same as **Hybrid**$_0$, except that the challenger guesses the edge $\overline{m}_{j^*,i^*}$ that is going to be missing in E' to be a path. When computing algorithm EncGS on step 4 (Definition 4), we introduce the following change: In step 4 in Figure 6 for $i = i^*$ and $j = j^*$, the random oracle H is simulated by lazy sampling. That is, a random bit string $(b_1, \ldots, b_\gamma)$ is sampled and the random oracle is programmed to provide this output on input $\mathsf{H}(\{c_i\}_{i=1}^N)$. Since there are $\frac{N(N+1)}{2}$ edges in G, the probability of guessing correctly the edge is $\frac{2}{(N(N+1)}$. Note that N is polynomial, which means that if an adversary has a non negligible advantage ϵ to win **Hybrid**$_0$, an adversary $\mathcal{B}$ using such adversary would have an advantage $\frac{2 \cdot \epsilon}{(N(N+1)}$, which is also non-negligible.
Hybrid$_2$: This hybrid is equivalent to **Hybrid**$_1$, except that we introduce a further modification to the implementation of algorithm EncGS. In particular for the indices $(b_1, \ldots, b_\gamma)$ and for $\overline{m}_{j^*,i^*}$ we compute $c^{(k)}_{j^*,i^*} \leftarrow \mathsf{WES.Enc}((\overline{pk}, \overline{m}_{j^*,i^*}), 0; \rho^{(k)}_{j^*,i^*})$ in line 3.b.ii. That is, we encrypt 0 instead of $r^{(k)}_{j^*,i^*}$.

Claim B.1. *Assume that WES encryption scheme is IND-CPA. Then,* **Hybrid**$_2$ *and* **Hybrid**$_1$ *are indistinguishable.*

Proof. **Hybrid**$_2$ and **Hybrid**$_1$ differ in the message encrypted in $c^{(k)}_{j^*,i^*}$ for all $k \in [1, \gamma]$. Consider now that between **Hybrid**$_2$ and **Hybrid**$_1$ we a have many subgames $\mathbf{Hybrid}^k_{sub}$, such that in each of them we only replace a single $c^{(k)}_{j^*,i^*}$ with an encryption of zero at a time. For instance in $\mathbf{Hybrid}^1_{sub}$ we only replace $c^{(1)}_{j^*,i^*}$, in $\mathbf{Hybrid}^2_{sub}$ we replace $c^{(1)}_{j^*,i^*}$ and $c^{(2)}_{j^*,i^*}$, until we reach **Hybrid**$_2$, in which we replace all $c^{(k)}_{j^*,i^*}$) Consider that the have an adversary $\mathcal{D}$ that has a non-negligible advantage to distinguish between subgames, $\mathbf{Hybrid}^k_{sub}$ and $\mathbf{Hybrid}^{k+1}_{sub}$. Adversary $\mathcal{B}$ can use adversary $\mathcal{D}$ to win WES IND-CPA with the following steps:

(1) $\mathcal{B}$ initiates the challenger and receives $\overline{pk}$.
(2) $\mathcal{B}$ computes $(pk, sk) \leftarrow \mathsf{KGen}(1^\lambda)$ and invokes $\mathcal{D}$ on input $(pk, \overline{pk})$
(3) $\mathcal{B}$ receives G from $\mathcal{D}$. Queries to $O\overline{\mathsf{Sig}}$ are forwarded to the challenger. Queries to $O\mathsf{Sig}$ are answered using sk.
(4) $\mathcal{B}$ computes $c \leftarrow \mathsf{EncGS}(sk, \overline{pk}, G)$. However, when computing $c^{(k)}_{j^*,i^*}$, note that there is only one that differs between $\mathbf{Hybrid}^k_{sub}$ and $\mathbf{Hybrid}^{k+1}_{sub}$. Hence, all of the ones that do not differ encrypt either 0 or $r^{(k)}_{j^*,i^*}$. We denote the one that differs as $c^{(k^*}_{j^*,i^*}$. To produce this ciphertext, $\mathcal{B}$ forwards the tuple $(\overline{m}_{j^*,i^*}, r^{(k^*)}_{j^*,i^*}, 0)$

to the challenger, and receives a ciphertext back, with is labeled as $c^{(k^*)}_{j^*,t^*}$. The rest of EncGS continues normally.
(5) $\mathcal{B}$ invokes $\mathcal{D}$ on input c and receives the tuple (σ^*, t^*).
(6) Finally, $\mathcal{D}$ sends a bit b with their guess if they were playing $\mathbf{Hybrid}^k_{sub}$ (b=0) or $\mathbf{Hybrid}^{k+1}_{sub}$ (b=1).
(7) $\mathcal{B}$ forwards the bit to the challenger.

$\mathcal{B}$ is a PPT algorithm and it perfectly simulates the subgames to $\mathcal{D}$. Since queries to $O\overline{\text{Sig}}$ are forwarded to the challenger, these two oracles are synchronized. Since the adversary wins, this means that $\overline{m}_{j^*,i^*} \notin E'$. Hence, $\overline{m}_{j^*,i^*}$ was not queried to $O\overline{\text{Sig}}$. The only difference between both subgames is the challenge ciphertext sent by the challenger. Therefore, the bit provided by $\mathcal{D}$ determines if $c^{(k^*}_{j^*,i^*}$ encrypts 0 or $r^{(k^*)}_{j^*,i^*}$, which can be used to break IND-CPA. However, this contradicts our assumption that IND-CPA hols, and the advantage of adversary $\mathcal{D}$ to distinguish both subgames is negligible. We can repeat this reasoning incrementally for each subgame, until reaching $\mathbf{Hybrid}_2$. Hence, we conclude that $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_1$ are indistinguishable.

□

$\mathbf{Hybrid}_3$: This hybrid is equivalent to $\mathbf{Hybrid}_2$, except that for all opened indices with $i = i^*$ and $j = j^*$ (i.e. $b_k = 0$ step 5 in Figure 6), we randomly sample each $s^{(k)}_{j^*,i^*}$ and set each $R^{(k)}_{j^*,i^*} = g^{s^{(k)}_{j^*,i^*}} / Y_{j^*,i^*}$. Since the distribution of $s^{(k)}_{j,i}$ and $R^{(k)}_{j,i}$ remains the same, $\mathbf{Hybrid}_3$ is indistinguishable from $\mathbf{Hybrid}_2$.
$\mathbf{Hybrid}_4$: Is equivalent to $\mathbf{Hybrid}_3$ except that for $i = i^*$ and $j = j^*$:

- Ciphertext ct_{j^*,i^*} (line 3.b.iv in Figure 6) is replaced with a random field element in $\mathbb{Z}_p$.
- Compute $\tilde{Y}_{j^*,i^*}$ (line 3.b.iii in Figure 6) as $g^{\text{ct}_{j^*,i^*}} / \text{x}_{i^*}$.
- Compute Y_{j^*,i^*} (line 3.b.iii in Figure 6) as $\tilde{Y}_{j^*,i^*} / \text{x}_{j^*}$.

Since the distribution of ct_{j^*,i^*}, $\tilde{Y}_{j^*,i^*}$ and Y_{j^*,i^*} is same, $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ are indistinguishable.
$\mathbf{Hybrid}_5$: Is equivalent to $\mathbf{Hybrid}_4$ except that after receiving the tuple (σ^*, t^*), the challenger checks if $(\tilde{Y}_{j,t^*}, \text{Extract}(\sigma^*, \widetilde{\sigma}_{j,t^*}, \text{x}) \notin \mathcal{R}$ for all $j < t^*$. If this check succeeds for all $j < t^*$, the challenger aborts the game.

Claim B.2. *Assume that the adaptor signature scheme achieves full extractability. Then, the challenger aborts with negligible probability in* $\mathbf{Hybrid}_5$.

Proof. Assume towards contradiction that $\mathcal{A}$ makes the challenger abort with no negligible probability in $\mathbf{Hybrid}_5$. We can build adversary $\mathcal{B}$, that breaks full extractability of the adaptor signature using $\mathcal{A}$ with the following steps:

(1) $\mathcal{B}$ initializes the challenger of full extractability and receives pk.
(2) $\mathcal{B}$ computes $(\overline{pk}, \overline{sk}) \leftarrow \overline{\text{KGen}}(1^\lambda)$ and invokes $\mathcal{A}$ on input $(pk, \overline{pk})$. Queries to $O\text{Sig}$ are forwarded to the challenger.
(3) $\mathcal{B}$ receives G from $\mathcal{A}$.
(4) $\mathcal{B}$ computes $c \leftarrow \text{EncGS}(sk, \overline{pk}, G)$. Note that in line 3.b.iv, $\mathcal{B}$ must compute $\widetilde{\sigma}_{j,i} \leftarrow \text{ADP.PreSig}(sk, m_i, \tilde{Y}_{j,i})$. If $i \neq i^*$ and $j \neq j^*$, sample $Y_{j,i}, y_{j,i}$ and send $m_i, Y_{j,i}$ to oracle $O\text{PreSig}$, to receive $\widetilde{\sigma}_{j,i}$. Otherwise, $\mathcal{B}$ samples random ct_{j^*,i^*} and computes $\tilde{Y}_{j^*,i^*} = g^{\text{ct}_{j^*,i^*}} / \text{x}_{i^*}$ and $Y_{j^*,i^*} = \tilde{Y}_{j^*,i^*} / \text{x}_{j^*}$. Then queries oracle $O\text{PreSig}$ on m_{i^*} and Y_{j^*,i^*} to receive $\widetilde{\sigma}_{j^*,i^*}$.
(5) $\mathcal{B}$ invokes $\mathcal{A}$ on input c and receives the tuple (σ^*, t^*).
(6) $\mathcal{B}$ sends (m_{t^*}, σ^*) to the challenger.

$\mathcal{B}$ is a PPT algorithm and that it perfectly simulates $\mathbf{Hybrid}_5$ to $\mathcal{A}$. If $\mathcal{A}$ makes the challenger to abort in $\mathbf{Hybrid}_5$, this means that: (i) m_{t^*} was not queried to $O\text{Sig}$; (ii) $\mathcal{A}$ did not query $O\overline{\text{Sig}}$ on message $\overline{m}_{j^*,i^*}$; (iii) the signature verifies. Furthermore, it also implies that the forgery σ^* has not been adapted from any of the presignatures that $\mathcal{B}$ gave to $\mathcal{A}$ since $(\tilde{Y}_{j,t^*}, \text{Extract}(\sigma^*, \widetilde{\sigma}_{j,t^*}, \text{x}) \notin \mathcal{R}$ for all $j < t^*$. Therefore, the forgery (m_{t^*}, σ^*) received by the challenger wins the full extractability game as well, with no negligible probability. However, this contradicts our assumption that full extractability holds, so we conclude that the challenger aborts in $\mathbf{Hybrid}_5$ with negligible probability.

□

So far, we have ensured that for the edge $\overline{m}_{j^*,i^*}$ that is missing in E' for it to be a path from m_0 to m_{t^*}, the adversary cannot obtain $r^{(k)}_{j^*,i^*}$ for any $k \in [1, \gamma]$, since we only encrypt 0. Furthermore, we sample ct_{j^*,i^*} so that $\mathcal{A}$ cannot obtain w_{i^*} from it. We also proved that the adversary cannot forge a signature, as it would break full extractability. Hence, the only option left of the adversary is to break the discrete logarithm for x_{i^*} to obtain w_{i^*} and be able to (i) either determine $\tilde{y}_{j^*,i^*}$ and adapt the presignature; or (ii) use w_{i^*} in the decryption of subsequent node.

Claim B.3. *Assume that the discrete logarithm is hard. Then, the probability that* $\mathcal{A}$ *wins* $\mathbf{Hybrid}_5$ *is negligible.*

Proof. Assume towards contradiction that $\mathcal{A}$ wins $\mathbf{Hybrid}_5$ with non negligible probability. We can build adversary $\mathcal{B}$, that breaks the discrete logarithm problem using $\mathcal{A}$ with the following steps:

(1) $\mathcal{B}$ initializes the challenger and receives x.
(2) $\mathcal{B}$ computes $(\overline{pk}, \overline{sk}) \leftarrow \overline{\text{KGen}}(1^\lambda)$ and $(pk, sk) \leftarrow \text{KGen}(1^\lambda)$.
(3) $\mathcal{B}$ invokes $\mathcal{A}$ on input $(pk, \overline{pk})$ and receives G.
(4) $\mathcal{B}$ can answer to queries to $O\text{Sig}$ and $O\overline{\text{Sig}}$.
(5) $\mathcal{B}$ computes $c \leftarrow \text{EncGS}(sk, \overline{pk}, G)$ as in $\mathbf{Hybrid}_5$. When $i = i^*$ and $j = j^*$ $\mathcal{B}$ computes $\tilde{Y}_{j^*,i^*} := g^{\text{ct}_{j^*,i^*}} / \text{x}$
(6) $\mathcal{B}$ invokes $\mathcal{A}$ on input c and receives the tuple (σ^*, t^*).
(7) $\mathcal{B}$ checks if $(\tilde{Y}_{j,t^*}, \text{Extract}(\sigma^*, \widetilde{\sigma}_{j,t^*}, \text{x}) \notin \mathcal{R}$ for all $j < t^*$. Since we have proven that $\mathcal{B}$ does not abort, at least for one $j < t^*$, it holds that $(\tilde{Y}_{j,t^*}, \text{Extract}(\sigma^*, \widetilde{\sigma}_{j,t^*}, \text{x}) \in \mathcal{R}$. For that j, compute $\text{w}_{t^*} := \text{ct}_{j,t^*} - \text{Extract}(\sigma^*, \widetilde{\sigma}_{j,t^*}, \text{x})$.
(8) $\mathcal{B}$ now has two scenarios. If $i^* = t^*$, then $\mathcal{B}$ sends w_{t^*} directly to the challenger as the forgery for x.
(9) Alternatively, if $i^* \neq t^*$, then $\mathcal{B}$ computes $\overline{\sigma}_{i^*,t^*} \leftarrow \overline{\text{Sig}}(\overline{sk}, \overline{m}_{i^*,t^*})$ and decrypts $r^{(k)}_{i^*,t^*} := \text{WES.Dec}(\overline{\sigma}_{i^*,t^*}, c^{(k)}_{i^*,t^*})$ and $y_{i^*,t^*} := s^{(k)}_{i^*,t^*} - r^{(k)}_{i^*,t^*}$. Then, computes $\text{w}_{i^*} := \text{ct}_{i^*,t^*} - \text{w}_{t^*} - y_{i^*,t^*}$ and sends w_{i^*} to the challenger as the witness corresponding to x.

$\mathcal{B}$ is a PPT algorithm and it perfectly simulates $\mathbf{Hybrid}_5$ to $\mathcal{A}$. If $\mathcal{A}$ wins with non-negligible probability, then it holds that $(\text{x}, \text{w}_{t^*}) \in \mathcal{R}$. Hence, $\mathcal{B}$ also wins the discrete logarithm problem with the same probability. This contradicts our assumption that the

discrete logarithm problem is hard, the probability to win $\mathbf{Hybrid}_5$ is negligible.

□

□

Proposition B.2. *If the adaptor signature achieves adaptability, the transitive tournament construction satisfies verifiability.*

Proof. Assume that $\mathcal{A}$ breaks the verifiability property for the transitive tournament construction. This implies that $\mathcal{A}$ outputs G, keys pk and $\overline{pk}$, a ciphertext c and index t. Then, for a subset of edges $E' \subset E$ with a path between $m_0 \in V$ and $m_t \in V$, $\mathcal{A}$ provides an attestations $\{\overline{\sigma}_{j,i}\}_{E'}$ for each $\overline{m}_{j,i} \in E'$. The adversary wins if three conditions are met:

- **Valid ciphertext:** Check that $\mathsf{VfEncGS}(c, pk, \overline{pk}, G) = 1$.
- **Valid attestation:** For each $\overline{\sigma}_{j,i}$ and $\overline{m}_{j,i} \in E'$, it holds that $\overline{\mathsf{Vf}}(pk, \overline{m}_{j,i}, \overline{\sigma}_{j,i}) = 1$.
- **Failed decryption.** Compute $\sigma_t := \mathsf{DecGS}(c, \{\overline{\sigma}_{j,i}\}_{E'})$ and obtain σ_t, such that: $\mathsf{Vf}(pk, m_i, \sigma_i) = 0$.

We now analyze each case individually.

First, we revise the checksin VfEncGS, which are below. They must pass for each $i \in [1, N]$ and each $j \in [0, i]$.

(1) Check that $g^{\mathsf{ct}_{j,i}} = Y_{j,i} \cdot \mathsf{x}_j \cdot \mathsf{x}_i$. This guarantees that the w_i is masked by the discrete log of $Y_{j,i}$ and w_j.
(2) Check that $\tilde{Y}_{j,i} = Y_{j,i} \cdot \mathsf{x}_j$. This guarantees that $\tilde{y}_{j,i}$ that is required to adapt the signature is computed as $y_{j,i} + \mathsf{w}_j$.
(3) Check that $\mathsf{ADP.PreVf}(pk, m_i, \tilde{Y}_{j,i}, \widetilde{\sigma}_{j,i}) = 1$. This implies that the pre-signature $\widetilde{\sigma}_{j,i}$ is valid.
(4) Check that $c_{j,i}^{(k)} = \mathsf{WES.Enc.Enc}((\overline{pk}, \overline{m}_{j,i}), r_{j,i}^{(k)}; \rho_{j,i}^{(k)})$ for opened indices. The indices are computed using one-way function H *after* computing $c_{j,i}^{(k)}$. The probability that, all ciphertexts in the open set $\mathcal{S}_{\mathsf{op}}$ pass the check, but all the ciphertexts in the unopened set $\mathcal{S}_{\mathsf{uop}}$ are malformed is $2^{-\gamma}$, in which γ is the security parameter of cut-and-choose. Hence, this probability is negligible.
(5) Check that$g^{s_{j,i}^{(k)}} = R_{j,i}^{(k)} \cdot Y_{j,i}$ for the unopened indices. This guarantees that the $s_{j,i}^{(k)} = r_{j,i}^{(k)} + y_{j,i}$ in the exponent.

Second, we now evaluate the different ways in which the decryption could fail and argue why they occur with negligible probability. We explicitly show below the arguments for $E' = \overline{m}_{0,t}$, but the same arguments could be repeated for any subset E' forming a path from m_0 to m_t.

- For all $k \in [1, \gamma]$ $r_{0,t}^{(k)} := \mathsf{WES.Dec}(\overline{\sigma}_{0,t}, c_{0,t}^{(k)})$ and get that $g^{r_{0,t}^{(k)}} \neq R_{0,t}^{(k)}$. This contradicts check 4) above, so this event does not happen.
- There exists one $r_{0,t}^{(k)} := \mathsf{WES.Dec}(\overline{\sigma}_{0,t}, c_{0,t}^{(k)})$ such that $g^{r_{0,t}^{(k)}} = R_{0,t}^{(k)}$, but it occurs that $y_{0,t} = s_{0,t}^{(k)} - r_{0,t}^{(k)}$ and $g^{y_{0,t}} \neq Y_{0,t}$. This contradicts check 5) above, so this event does not happen.
- We have $y_{0,t}$ such that $g^{y_{0,t}} = Y_{0,t}$ but for $\tilde{y}_{0,t} = y_{0,t} + \mathsf{w}_0$, we have that $g^{\tilde{y}_{0,t}} \neq \tilde{Y}_{0,t}$. This contradicts check 2) above, so this event does not happen.
- We have $\tilde{y}_{0,t}$ such that $g^{\tilde{y}_{0,t}} = \tilde{Y}_{0,t}$, but for $\mathsf{w}_i = \mathsf{ct}_{0,t} - \tilde{y}_{0,t}$, we have that $(\mathsf{x}_t, \mathsf{w}_t) \notin \mathcal{R}$. This contradicts check 1) above, so this event does not happen.
- We have $\tilde{y}_{0,t}$ such that $g^{\tilde{y}_{0,t}} = \tilde{Y}_{0,t}$, but for $\sigma_t := \mathsf{ADP.Adapt}(\widetilde{\sigma}_{0,t}, \tilde{y}_{0,t})$, we have that $\mathsf{Vf}(pk, m_t, \sigma_t) = 0$. Since check 3) passed, this means that the presignature $\widetilde{\sigma}_{0,t}$ verifies for m_t, $\tilde{Y}_{0,t}$ and pk. Therefore, this contradicts our assumption that the adaptor signature scheme satisfies adaptability. Hence, we conclude that $\mathsf{Vf}(pk, m_t, \sigma_t) = 1$

□

C Construction for Linear Graph and Proof

Proposition C.1. *Assume that the adaptor signature scheme achieves full extractability, the WES encryption scheme is IND-CPA and the discrete log problem is hard. Then, the linear construction satisfies graph enforcement.*

Proof. This proof works exactly as Proposition B.1. Towards a contradiction, we present a series of hybrids until we reduce the last one to hardness of the discrete logarithm problem. $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ are identical to Proposition B.1. Since the edges now are represented by a linear number of messages $\overline{m}_{i-1,i} \in E$, then the probability of guessing the edge missing in E' is now $1/N$.
$\mathbf{Hybrid}_2$: This hybrid is equivalent to $\mathbf{Hybrid}_1$, except that we modify further algorithm EncGS. In particular for the indices $(b_1, \ldots, b_\gamma)$ and for each $\overline{m}_{i^*\text{-}1,i^*}$, if $b_k = 0$, we compute:

$$c_{i^*}^{(k)} \leftarrow \mathsf{WES.Enc}((\overline{pk}, \overline{m}_{t^*\text{-}1,i^*}), 0; \rho_{i^*}^{(k)})$$

in line 3.b. That is, we encrypt 0 instead of $r_{i^*}^{(k)}$.

Claim C.1. *Assume that WES is IND-CPA. Then,* $\mathbf{Hybrid}_2$ *and* $\mathbf{Hybrid}_1$ *are indistinguishable.*

Proof. This proof works exactly as Claim B.1, but replacing step 4) with the following:

$\mathcal{B}$ computes$c \leftarrow \mathsf{EncGS}(sk, \overline{pk}, G)$. When computing $c_{i^*}^{(k)}$, note that only one differs between $\mathbf{Hybrid}_{sub}^{k}$ and $\mathbf{Hybrid}_{sub}^{k+1}$. Hence, all of the ones that do not differ encrypt either 0 or $r_{i^*}^{(k)}$. We denote the one that differs as $c_{i^*}^{(k^*)}$, for index $k = k^*$. To produce this ciphertext, $\mathcal{B}$ forwards the tuple $(\overline{m}_{i^*\text{-}1,i^*}, r_{i^*}^{(k^*)}, 0)$ to the challenger, and receives a ciphertext back, with is labeled as $c_{i^*}^{(k^*)}$.

□

$\mathbf{Hybrid}_3$: This hybrid is equivalent to $\mathbf{Hybrid}_2$, except that for all opened indices with $i = i^*$ (i.e. $b_k = 0$ step 5 in Figure 12), we randomly sample each $s_{i^*}^{(k)}$ and set each $R_{i^*}^{(k)} = \frac{g^{s_{i^*}^{(k)}}}{Y_{i^*} \cdot R_1^{(k)} \cdot, \ldots, R_{i^*\text{-}1}^{(k)}}$.
Since the distribution of $s_{i^*}^{(k)}$ and $R_{i^*}^{(k)}$ remains the same, $\mathbf{Hybrid}_3$ is indistinguishable from $\mathbf{Hybrid}_2$.
$\mathbf{Hybrid}_4$: Is equivalent to $\mathbf{Hybrid}_3$ except that for $i = i^*$:

- Each ciphertext ct_{i^*} (line 3.d in Figure 12) is replaced with a random field element in $\mathbb{Z}_p$.
- Compute $\tilde{Y}_{i^*}$ (line 3.c in Figure 12) as $g^{\mathsf{ct}_{i^*}}/\mathsf{x}_{i^*}$.
- Compute Y_{i^*} (line 3.c in Figure 6) as $\tilde{Y}_{i^*}/\mathsf{x}$.

Public parameters: $(\mathbb{G}, g, q, \gamma, \mathsf{H}, N)$

$c \leftarrow \mathsf{EncGS}(sk, \overline{pk}, G := (V, E))$:

(1) Initialize $\mathcal{S}_{\mathsf{op}} := [\cdot]$; $\mathcal{S}_{\mathsf{uop}} := [\cdot]$ // Initialize opened and unopened sets
(2) Compute $\{(\mathsf{w}_i, \mathsf{x}_i)\}_{i=0}^{N} \leftarrow \mathsf{createR}^{N+1}(1^\lambda)$ // Prepare witnesses to enable graph navigation.
(3) Repeat for each $i \in [1, N]$: // Iterate over all vertices in the graph
  (a) Parse $m_i \in V$ and $\overline{m}_{i\text{-}1,i} \in E$
  (b) Repeat for each $k \in [1, \gamma]$:
    – Sample $r_i^{(k)} \xleftarrow{\$} \mathbb{Z}_q$; and set $R_i^{(k)} := g^{r_i^{(k)}}$
    – Compute $c_i^{(k)} \leftarrow \mathsf{WES.Enc}((\overline{pk}, \overline{m}_{i\text{-}1,i}), r_i^{(k)}; \rho_i^{(k)})$; where $\rho_i^{(k)}$ are the random coins.
  (c) Sample $y_i \xleftarrow{\$} \mathbb{Z}_q$, compute $\tilde{y}_i = \mathsf{w}_0 + y_i$ and set $Y_i := g^{y_i}$ and $\tilde{Y}_i := g^{\tilde{y}_i}$ // Create $\tilde{y}_i$
  (d) Compute $\mathsf{ct}_i := \mathsf{w}_i + \tilde{y}_i$ and $\widetilde{\sigma}_i \leftarrow \mathsf{ADP.PreSig}(sk, m_i, \tilde{Y}_i)$ // Encrypt signature & witness
  (e) Set $\Sigma_i := (\widetilde{\sigma}_i, \tilde{Y}_i, Y_i)$
  (f) Set $c_i := (\{c_i^{(k)}\}_{k=1}^{\gamma}, \{R_i^{(k)}\}_{k=1}^{\gamma}, \Sigma_i, \mathsf{ct}_i)$
(4) Compute $(b_1, b_2, ..., b_\gamma) := \mathsf{H}(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0)$ // Determine indices to open
(5) Repeat for each $k \in [1, \gamma]$:
  – If $b_k = 1$, for all $i \in [1, N]$ $\mathcal{S}_{\mathsf{op}}[k, i] := (r_i^{(k)}, \rho_i^{(k)})$ // Open $c_i^{(k)}$
  – If $b_k = 0$, for all $i \in [1, N]$ set $s_i^{(k)} := r_1^{(k)} + \ldots + r_i^{(k)} + y_{j,i}$ and $\mathcal{S}_{\mathsf{uop}}[k, i] := s_i^{(k)}$
(6) Return $c := (\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0, \mathcal{S}_{\mathsf{op}}, \mathcal{S}_{\mathsf{uop}})$

$b := \mathsf{VfEncGS}(c, pk, \overline{pk}, G := (V, E))$:

(1) Parse $(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0, \mathcal{S}_{\mathsf{op}}, \mathcal{S}_{\mathsf{uop}}) \leftarrow c$
(2) Compute $(b_1, b_2, ..., b_\gamma) := \mathsf{H}(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0)$ // Determine indices to open
(3) Repeat for each $i \in [1, N]$: // Iterate over all vertices in the graph
  (a) Parse $m_i \in V$ and $\overline{m}_{i\text{-}1,i} \in E$ and $(\{c_i^{(k)}\}_{k=1}^{\gamma}, \{R_i^{(k)}\}_{k=1}^{\gamma}, \Sigma_i, \mathsf{ct}_i) \leftarrow c_i$
  (b) Parse $(\widetilde{\sigma}_i, \tilde{Y}_i, Y_i) \leftarrow \Sigma_i$
  (c) Check that $g^{\mathsf{ct}_i} = Y_i \cdot \mathsf{x}_0 \cdot \mathsf{x}_i$ and $\tilde{Y}_i = Y_i \cdot \mathsf{x}_0$ // Check ciphertext ct_i
  item Check that $\mathsf{ADP.PreVf}(pk, m_i, \tilde{Y}_i, \widetilde{\sigma}_i) = 1$. // Check presignature $\widetilde{\sigma}_i$
  (d) Repeat for each $k \in [1, \gamma]$:
    – If $b_k = 1$, for $(r_i^{(k)}, \rho_i^{(k)}) \leftarrow \mathcal{S}_{\mathsf{op}}[k, i]$, check $c_i^{(k)} = \mathsf{WES.Enc}((\overline{pk}, \overline{m}_{i\text{-}1,i}), r_i^{(k)}; \rho_i^{(k)})$ // Check $c_i^{(k)}$
    – If $b_k = 0$, for $s_i^{(k)} \leftarrow \mathcal{S}_{\mathsf{uop}}[k, i]$, check $g^{s_i^{(k)}} = R_1^{(k)} \cdot \ldots \cdot R_i^{(k)} \cdot Y_i$ // Check $s_i^{(k)}$
  (e) If any check fails abort and return 0
(4) Return 1

$\sigma_t := \mathsf{DecGS}(c, \{\overline{\sigma}_{j,i}\}_{E'})$:

(1) Parse $(\{c_i\}_{i=1}^{N}, \{\mathsf{x}_i\}_{i=0}^{N}, \mathsf{w}_0, \mathcal{S}_{\mathsf{op}}, \mathcal{S}_{\mathsf{uop}}) \leftarrow c$ and $(\overline{\sigma}_{0,1}, ..., \overline{\sigma}_{t-1,t}) \leftarrow \{\overline{\sigma}_{j,i}\}_{E'}$
(2) For $k \in [1, \gamma]$ do: // Compute $r_{ac} = r_{j+1}^{(k)} + \ldots + r_i^{(k)}$
  (a) Set $r_{ac} = 0$ for $i \in [1, t]$ do:
    (i) Compute $r_i^{(k)} := \mathsf{WES.Dec}(\overline{\sigma}_{i\text{-}1,i}, c_i^{(k)})$ and set $r_{ac} = r_i^{(k)} + r_{ac}$
  (b) If $g^{r_{ac}} \neq R_{m+1}^{(k)} \cdot \ldots \cdot R_l^{(k)}$, next k, else go to 3) // At least one r_{ac} will be correct
(3) Get c_t from $\{c_i\}_{i=1}^{N}$ and parse $(\{c_t^{(k)}\}_{k=1}^{\gamma}, \{R_t^{(k)}\}_{k=1}^{\gamma}, \Sigma_t, \mathsf{ct}_t) \leftarrow c_t$
(4) Get ct_t, $\widetilde{\sigma}_t$ and Y_t
(5) For any $k \in [1, \gamma]$ and $s_t^{(k)} \in \mathcal{S}_{\mathsf{uop}}[k, t]$, do:
  (a) Compute $y_t := s_t^{(k)} - r_{ac}$ and $\tilde{y}_t = y_t + \mathsf{w}_0$
  (b) Compute $\sigma_t := \mathsf{ADP.Adapt}(\widetilde{\sigma}_t, \tilde{y}_t)$
  (c) Return σ_t

Figure 12: VGES **construction for a linear graph.**

Since the distribution of ct_{i^*}, $\tilde{Y}_{i^*}$ and Y_{i^*} is same, $\mathbf{Hybrid}_3$ and $\mathbf{Hybrid}_4$ are indistinguishable.

$\mathbf{Hybrid}_5$: Is equivalent to $\mathbf{Hybrid}_4$ except that after receiving the tuple (σ^*, t^*), the challenger checks if $(\tilde{Y}_{t^*}, \mathsf{Extract}(\sigma^*, \widetilde{\sigma}_{t^*}, \mathsf{x}) \notin \mathcal{R}$. If this check succeeds, the challenger aborts the game.

Claim C.2. *If the adaptor scheme achieves full extractability, the probability to abort in* $\mathbf{Hybrid}_5$ *is negligible.*

Proof. This proof works exactly as the proof for Claim B.2. □

Claim C.3. *Assume that the discrete logarithm is hard. Then, the probability that* $\mathcal{A}$ *wins* $\mathbf{Hybrid}_5$ *is negligible.*

Proof. This proof works exactly as the proof for Claim B.3. □

□

Proposition C.2. *If the adaptor signature achieves adaptability, the linear construction satisfies verifiability.*

Proof. This proof works exactly as Proposition B.2. □

D Extensions to our protocol

D.1 Privacy

Even though we only use basic transactions and no scripts, user profiling can still be achieved. Consider an adversary monitoring blockchains $\mathcal{L}$ and $\widehat{\mathcal{L}}$. The transactions $\widehat{m}_{j,i}$ submitted by the client to $\widehat{\mathcal{L}}$ may share the same sender, $\widehat{pk}_C$. Using this information, the adversary collects transactions with $\widehat{pk}_C$ as the sole sender and uses address clustering techniques (e.g., [33, 38, 39]) to identify which receivers are controlled by the same entity, namely, the bank. The adversary labels these transactions as $(\widehat{m}_1, \ldots, \widehat{m}_N)$, where $\widehat{m}_1$ is the first transaction to appear in the blockchain and $\widehat{m}_N$ the last. By adding the transaction amounts and examining $\mathcal{L}$ for transactions occurring shortly before $\widehat{m}_1$ and shortly after $\widehat{m}_N$, the adversary may identify the collateral-locking and collateral-withdrawal events. Hence, the adversary may be able to learn that: (i) there could be a contract involving two ledgers; (ii) requires N transactions on $\widehat{\mathcal{L}}$; (iii) the total amount involved. We discuss a few strategies to reduce the information leaked with our approach.

Escrow in a payment channel. In our description of the escrow address on blockchain $\mathcal{L}$, we considered an on-chain escrow account. However, if client and bank already share a payment channel, the escrow can be implemented off-chain, within that channel. In this case, collateral-locking and collateral-withdrawal occur off-chain, preventing the adversary from linking transactions on $\mathcal{L}$ to events on $\widehat{\mathcal{L}}$. Hence, the adversary might be able to cluster the transactions that occur in $\widehat{\mathcal{L}}$, but not link them to an event on $\mathcal{L}$.

Employ coin mixing for repayments. In the basic setting described above, an observer could scan $\widehat{\mathcal{L}}$ for transactions originating from the same sender, $\widehat{pk}_C$, and identify those directed to the same bank, thereby linking repayments. To mitigate this, the sender can issue repayment transactions through a coin mixing protocol (e.g., those in [23, 37, 43], which require only basic transactions). Coin mixing protocols involve three parties, the sender, the receiver a the mixer. At a high level, a set of senders transfer α coins to the mixer, and a set of receivers withdraw α coins in a randomized order. The key privacy guarantee is unlinkability, which prevents any observer, including the mixer, from determining which sender corresponds to which receiver. As a result, if the client performs repayments through coin mixing, an adversary may still observe the total amount received by the bank, but cannot reliably determine which transactions originated from the client.

Hide the repayment amounts. The client and bank can choose for $\widehat{\mathcal{L}}$ a blockchain that conceals the transaction amount. For instance, they could use Zether [6] or a Bitcoin side-chain supporting confidential assets [36]. In this case, during the setup stage, the client and bank agree not only on the amounts α_i, but also on a commitment C_i and an opening p_i for each amount. Repayment options are then defined as $\overline{m}_{i\text{-}1,i} := (pk_{\mathcal{B}_i}, C_i)$. While the adversary can still use address clustering techniques to identify transactions with a single sender and the same receiver entity, linking these transactions to the corresponding transactions on $\mathcal{L}$ becomes more difficult, as the amounts are hidden.

Hide sender and receiver. Alternatively, the client and bank can choose for $\widehat{\mathcal{L}}$ a blockchain that hides both transaction amounts and the link between the sender and the receiver. An example could be Zcash [4], in which the public key of the receiver does not appear in shielded transactions. During setup, bank and client agree on the coin commitments of the coins that the bank should receive in each repayment, which we denote as C_i. In Zcash, the oracle would attest the coin commitments created in each transaction. In this scenario, the adversary would not be able to infer information about the loan from the information in the blockchain alone, and would need to rely on other alternatives, such a side-channel attacks [46].

D.2 Dynamic Exchange Rate: Liquidation

The exchange rate between $\mathcal{L}$ and $\widehat{\mathcal{L}}$ would fluctuate in a real environment. This means that the value of the cryptocurrency used as collateral may decrease. If the collateral becomes worthless, the client has no incentive to repay the loan, as the loan coins are more valuable. Meanwhile, the bank is left with a devalued collateral locked in the escrow account. To mitigate this risk, smart-contracts allow the bank to liquidate the collateral when its price falls to a critical price P. The critical price P is computed such that, if the bank sells the collateral at this price, they fully recover the value of the loan. Our protocol can be extended to support liquidation through the following steps.

Oracle assumptions. The oracle periodically attests the exchange rate r by: (1) computing r between $\mathcal{L}$ and $\widehat{\mathcal{L}}$ and rounding it to the closest integer; (2) creating message $\overline{m}_{p_i} = (id, r)$ with timestamp id; (3) signing $\overline{\sigma} \leftarrow \mathrm{Sig}(\overline{pk}_O, \overline{m}_{p_i})$; and (4) publishing $(\overline{m}_{p_i}, \overline{\sigma})$. We assume that the timestamps are such that if $P_1 > P > P_2$, attesting P_1 and P_2 implies that P was also attested in between.

Setup modification. The bank and client agree on a critical price P and a set of messages $\{(id, P)\}$ for each timestamp from the contract starting time until the escrow timeout T. They also agree on the graph $G = (V, E)$. The client computes with EncSig [31] the liquidation ciphertext, which encrypts a signature on message m_0 (i.e., the initial termination state) that can only be decrypted if the bank has an attestation on any of the messages $\{(id, P)\}$. The liquidation ciphertext is sent to the bank for verification, i.e. VfEncSig [31]. Upon successful verification, the protocol continues as in Figure 4, with the bank computing c with VGES.

Liquidation. If the oracle attests $\overline{m}_{p_i} \in \{(id, P)\}$, the bank computes DecSig of [31] on the ciphertext sent by the client to obtain $\sigma_{C,0}$. The bank then signs m_0 to get $\sigma_{\mathcal{B},0}$ and submits m_0 to $\mathcal{L}$.

Security intuition. Security relies on the one-wayness and verifiability of VWETS [31]. The bank cannot obtain $\sigma_{C,0}$ without an oracle attestation, and the client cannot produce a malformed ciphertext that verifies. In our protocol, if the client sees that the oracle has attested to a price P' close to the critical price P, they may terminate the contract by submitting the latest state m_t they have unlocked. In this case, the bank only claims the portion of the collateral corresponding to the outstanding debt. Liquidation thus serves as a last-resort mechanism that the bank can invoke if the client becomes unresponsive.

Overhead. The liquidation ciphertext requires the same computation and communication complexity as a single state in Figure 12. Hence, we conclude that the computational overhead would be of 145ms and the communication overhead of 84kB.

D.3 Dynamic Exchange Rate: Margin Call

Another common mechanism in smart-contract lending is the margin call. When the value of the collateral approaches the critical price P, the bank may invite the client to increase the collateral by locking additional coins. If the client provides more collateral, the collateral-to-loan ratio increases, signaling a stronger commitment to repay the loan. Margin calls also result in a new critical price P', that enables the bank to recover the value of the loan by selling the original collateral and the new collateral provided by the client at price P'. Our protocol can be extended to support margin calls with the following steps:

Setup modification. Instead of computing a single c with VGES, the bank and client agree on a set of potential margin calls that may occur in the contract duration. We now discuss the case for a single margin call, but the process is equivalent for more. The bank and client agree on two graphs, denoted by the graphs $G = (V, E)$ and $G_{mc} = (V_{mc}, E_{mc})$. The escrow address of G is esc and of G_{mc} is esc_{mc}. If the collateral for G is κ, the collateral in G_{mc} is $\kappa_{mc} = \kappa + \kappa_{mc}$, in which κ_{mc} represent the coins that the client provides during the margin call. Regarding liquidation, the bank and the client agree on two liquidation prices, P for G and P_{mc} for G_{mc}. For graph G_{mc}, we set the edges to be the same as in G, $E_{mc} := E$. Therefore, the events that trigger state updates are equivalent in both graphs. However, the actual termination states are different, since one graph has more collateral than the other. Hence in the margin call graph, the transactions m_i spend from esc_{mc} and distribute κ_{mc} coins instead of esc and κ. Given G, P, G_{mc}, and G_{mc}, we follow the steps in Appendix D.2 to produce the liquidation ciphertexts for P and P_{mc}; and run algorithm EncGS to compute c and c_{mc}.

Margin call transaction. The client creates the margin call transaction m_{mc} that transfers: (a) κ coins from esc to esc_{mc}; and (b) κ_{mc} coins from pk_C to esc_{mc}. To be appended in the blockchain, m_{mc} requires signatures from the bank and the client. The bank signs m_{mc} and gives the signature $\sigma_{\mathcal{B},mc}$ to the client. To submit the transaction to $\mathcal{L}_0$, the client only needs to sign it and add their signature to that of the bank.

Security intuition. The ciphertexts c and c_{mc} are secure as they satisfy the security of VGES. Similarly, the liquidation ciphertexts for P and P_{mc} are secure for the same reasons discussed in Appendix D.2. Once the setup is completed, the client has the margin call transaction m_{mc} signed by the bank, so the bank cannot prevent the client to submit it. Therefore, if the price approaches P, the client can either: (i) submit m_{mc} to $\mathcal{L}$, thereby activating the margin call; (ii) submit the latest unlocked state m_t in graph G to $\mathcal{L}$; or (iii) do nothing, allowing the bank to execute collateral liquidation.

Overhead. The margin call mechanism requires computing a fresh set of graph ciphertexts for each enabled margin call. Consequently, both computation and communication overhead scale linearly with the number of margin calls agreed upon by the bank and client during setup. We believe, however, that there is room for optimization: the graphs associated with successive margin calls contain substantial redundancy, suggesting that tailored techniques could significantly reduce both computational and communication costs.