HyperFridge Whitepaper

Enabling trustless atomic swaps between crypto assets and fiat backends

Walter Strametz, walter.strametz@gmail.com

February 2023, Version 0.0.1

1 Abstract

Many of today's regulations aim at reducing "asymmetric information" which occurs when one party to an economic transaction possesses greater material knowledge than the other party. For example companies need to follow a regime of transparency requirement in order to get listed on a stock exchange. For blockchain protocols transparency is usually given due to its open nature – anyone has access to same information because transaction are public or anyone can check code online on platforms like Github – for larger projects public scrutiny is always given. But as soon as blockchain tabs into traditional finance, it enters the realm of asymmetric information. Just think of fully pegged stablecoins – we need additional (off-chain) information on the pegging of a token – the issues needs to be audited or needs to publish account statements on his own will.

We propose a privacy preserving system which reduces the gap of "asymmetric information" between the Crypto and the Fiat world in a cryptographically proven and therefore trustless manner. It creates the very foundation to expand the reach decentralized business models and smart contract into the world of traditional finance. Use cases might be to buy and sell tokens via a decentralized exchange like Uniswap based on Fiat payments. Or anyone can buy an NFT via a Fiat payment like you would buy a pair of sneakers on an online webshop - instantly, without credit risk or an intermediary to change Fiat into Eth.

"*HyperFridge* " is a cryptographically verifiable system building on top of the open-banking standard ISO20022, which entangles signatures of a bank and its client into SNARKs to prove transaction inclusion, balances and to trigger wire-transfers. As a result, we are able to provide a infrastructure for swaps between fiat and digital assets, without using intermediaries like an exchange. With that, any holder of a bank account can "connect" his own fiat based account to any blockchain or smart contract in a decentralized und trustless manner. For many real-world use cases we remove the need of a central party for exchanging value between the fiat and blockchain world. Looking at B2B standards like FIX, we also think that this idea is also expandable to traditional assets like stocks or any other electronical tradable financial product.

2 Introduction *HyperFridge*

Blockchain enables secure and trustless transactions of value between parties. However, despite a growing ecosystem, cryptocurrencies continue to operate in complete isolation from traditional finance: blockchain protocols do *not* provide a secure *or* trustless way to on- or off-ramp values with the finance industry. Connecting blockchain ecosystem (or smart contracts) with the banking system remains an open challenge.

Centralized exchanges are the only bridges between traditional and decentralized finance. These services require trust and therefore undermine the very nature of the decentralization, making them vulnerable to attacks. Although decentralized exchanges (DEXs) remove the need to trust centralized intermediaries for blockchain transfers, they still require that value has already moved on-chain - thus do not provide decentralized bridge into traditional finance.

Therefore we propose a system called *HyperFridge* , which acts as a bridge between traditional and decentralized finance, enabling account holders to connect a blockchain ledger with their bank account, in a trustless manner. *HyperFridge* replicates balance and transaction data of a bank account and triggers wire transfers to external bank accounts - publicly verifiable but keeping data privacy by using zero knowledge proofs.

An instance of a *HyperFridge* node provides a bridge which:

1. Works on the fiat-to-blockchain (on-ramp) direction as an oracle towards a bank account with a ZK-SNARK setup including the e-banking client and server certificates in the proofs.

2. Works in the direction of blockchain-to-fiat (off-ramp) by using on-chain events to trigger wire-transfers from the pegging a bank-account.

3. But delegates higher level logic (smart contracts) out of scope, which enables to implement use cases like stablecoins or fiat-bridges with asset-locking [21, 25] for immediate settlement.

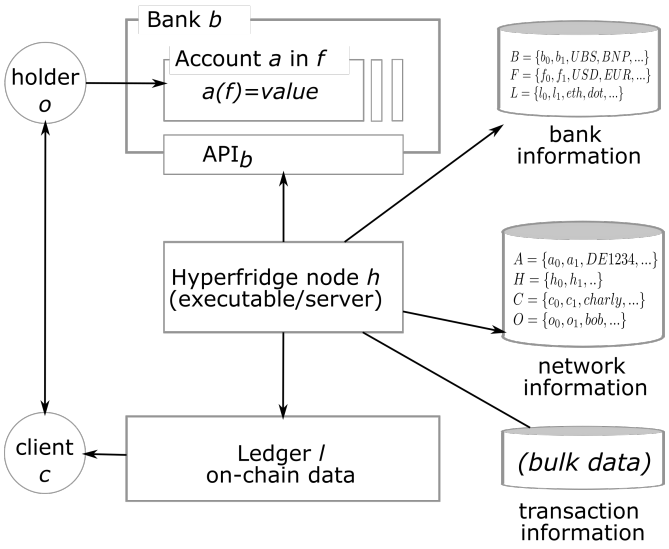
Today the crypto industry's call for a stablecoin provider needs external audits to proof it has the reserves for the Token it is issuing. As a use case of among many, *HyperFridge* is able to invalidate this assumption: Anyone can issue fully pegged stablecoins based on an pegging bank account on a zero-trust basis, no need for external auditing who will look at the pegging ratio. A proof of liability (pegging ratio) is "built-in". This means that the system can prove the pegging bank balance without trusting the issuing counterparty - with *HyperFridge* we can apply trustless logic based on cryptographic proofs of off-chain fiat account data. This property is key for creating decentralized trustless bidirectional fiat bridges.

3 Overview

3.1 Definitions

As fiat backends we define standardized APIs which provide access to bank accounts. These APIs allow to read transactions and balances its its finality status, and support creating new transactions to wire-transfer funds to other bank accounts. A bank account is

Figure 1: Conceptual overview of definitions and its relations



represented by A, its currency by F and the fiat balance of a specific account as *a(f)* (e.g. CHF 400 of account CH123490). We define:

- *(pegging) bank account*, the bank account *A* that backs *a(f)* using fiat currency f at bank b
- *(fiat) value*, the units of the backing cryptocurrency used to generate the asset *a(f)* – e.g. $a(f)_{bo} = 400 \Rightarrow 400$ in currency f on bank account a at bank b with account holder o.
- *account holder*, the owner (beneficiary) *o* of *a(f)*, who has control of the private keys to access and use the APIs of the fiat backend.
- *API (bank gateway)*, the actual APIs (endpoints) to access account a of currency f granted to *account holder o* via API_{bfao} in short API.
- As h we define an instance of *HyperFridge* (a node) which operates between a bank account a, which is controlled by an API provided by a bank. Using private keys of the *account holder* this program downloads latest transactions and balances, uploads payment instructions and generates zero-knowledge proofs using private API keys and bank signatures as secret input for the proofs.
- The *ledger* l: The ledger which stores the state of a bank account, retrieved by h about on- and off-ramping event. On top of this data structure, smart contracts can issue e.g. fiat based asset tokens like stablecoins. It is important to point out, that the system model for *HyperFridge* does not issue fiat tokens (stablecoins) in the first place. It is about triggering and mirroring state changes of a bank account on a ledger. We want to create a minimum technical scope which supports connectivity to the banking system, without introducing smart sontracts or Tokens.
- The *(fiat) client* is the entity (person) who wants to want to on- or off-ramp fiat funds to change state on a blockchain ledger, e.g. to buy tokens with a fiat payment.

3.2 System Goals

To achieve a trust-less execution environment for both including blockchain and banking network, we want to achieve following properties:

- *Auditability and data availability:* A user participating in a transaction can access and verify transaction data to audit the operation and to detect protocol failures, also for a later point in time.
- *Consistency and completeness:* Balance $a(f)$ and all transactions are reflected consistently and without gaps in their occurring order. Wrong balances or missing transactions must lead to an inconsistent state.
- *Liveness:* Any user can issue, transfer and swap fiat with tokens, without requiring a third party other than a bank. Liveness only relies in the general properties of blockchain ledger l and the banking network itself - not the issuer or controller of the pegging bank account.
- *Atomicity:* On- and off-ramping of fiat should enable atomic swaps between fiat (traditional) and digital assets (tokens). Users can at best instantly swap fiat currencies with tokens other assets on a ledger - it either fails or succeeds withing a certain period of time.
- *Blockchain-agnostic:* Developers should be able to select the blockchain protocol that best serves a specific use case, deploy separate instances of a *HyperFridge* node which can connect to that specific protocol but using shared data and proofs. Ideally we have two protocols supporting the *HyperFridge* system goals - currently we aim an EVM based protocols and Polkadot where we have already implemented fiat bridges based on standard banking APIs [12, 23].

3.3 System Events

The goal is to capture balances, mirror transactions and trigger wire-transfers on bank accounts and later reflect its settlement on a blockchain ledger - in a trustless manner. On other words - it is about read and write access to a bank account, capturing the data on a ledger and provide cryptographic proof for correctness of balance and included transactions.

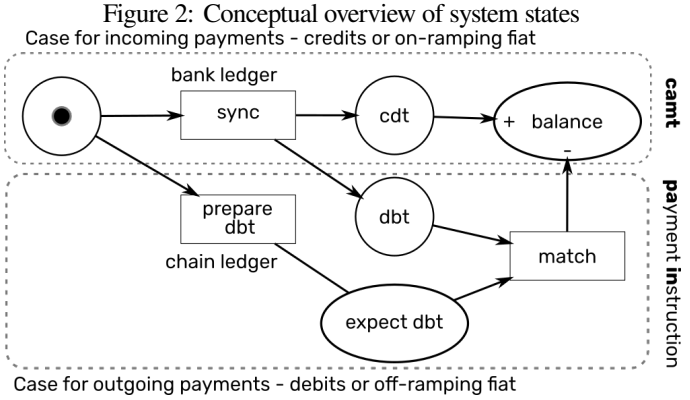
We define following events in the system which trigger activity through state changes on a bank account or the ledger (2):

- *Account update:* The state of an account changes $a \rightarrow a'$ by an incoming payment (credit/on-ramp), an outgoing payment (debit/off-ramp) or by a technical booking triggered by the bank itself for interest (credit) or fees (debit). In any case we have a counterparty which is either the client c or the bank b, but we ignore the second case for the sake of simplicity. The role of *HyperFridge* is to mirror the state change on the ledger $l \rightarrow l'$.
- *Prepare debit:* In a bidirectional bridge we want to off-ramp fiat which means to be able to initiate the *prepare phase* of a wire-transfer (debit) directly on-chain. We can use a smart contract for sending an off-ramp signal (e.g. burning coins as implemented in [12]), send funds to a specific burn address (like 0x0) or just call permissioned extrinsics [14]. In the end it can be any state change on the ledger ($l \rightarrow l'$) which lead to a state transaction of $a \rightarrow a'$. Finality can only be given if we detect the debit (the a') on the bank account in the *account update*. Therefore we are proposing a classical *two phase commit* with a prepare phase and a commit phase.

Data connected to the events:

- *Account update*: The tuple $acccountUpdate_{boi} = balance, (txId, cdtDbt, iban, amount)^*$: represents the i^{th} account updated of account holder o at bank b. For example

Figure 2: Conceptual overview of system states



some client c transfers funds to bank b to the bank account of o, whereas $cdtDbt$ flags incoming ("CDT") or outgoing ("DBT") transactions.

- *Prepare debit*: The tuple $updateLedger_l(iban, amount)$ is initiated on-chain and triggers the first phase to off-ramp fiat as described above.

An example: Alice wants to buy a real estate property from Bob and wire-transfers 5 USD to Bob's Bank account as a down-payment. Bob runs a *HyperFridge* node h which reads transaction of his own bank account a. The payment gets picked up via the API and mirrored on the ledger l, which acts as a proof for the downpayment. The downpayment can now be processed by a smart sontract.

3.4 Synching the two ledger systems

The term "settlement finality" in traditional finance defines that a transfer order is finally settled, regardless of whether the sending participant has become insolvent or transfer orders have been revoked in the meantime [20]. Along with ISO20022 related initiatives, the European Central Bank works [9] on harmonization, the settlement finality in payment, and securities settlement systems [19]. Also part of implementing ISO20022 messages are instant payments, which will in future allow immediate (user friendly) fiat-to-token swaps.

Many banks will (in EU they must) support ISO20022 APIs and its properties, but we need a "higher power" to decide if the counterparty is actually a real bank and therefore trust its public certificates. For a productive system we would see an organization or DAO to be responsible to map current state of (bank) audits into the *HyperFridge* network (e.g. implemented through validators).

For a proof on concept and as initial system, we would hand-pick selected banks and assume intraday finality through the "booked" status of bank transactions in a "Camt.053" message [17]. The following algorithm describes the update of the ledger $l \rightarrow l'$ based on the data retrieved from the bank backend API for account a of holder o.

Variables starting with caption (e.g. $tx.Amount$) are elements of Camt.053 message. Within this message, each XML node ($< Entry > ... < /Entry >$) is referred as tx (transaction). The blockchain ledger is refered as l:

Algorithm 1 Synching a with l through API

```
1:  procedure SYNCHRONIZE
2:      txns = API_{b,o}.getCamt053_a()                          ▷ s1
3:      for all tx in txns do
4:          if tx.CredDeb == "CRT" then                          ▷ s2
5:              l.txsIn = l.txsIn ∪ tx
6:              l.balance += tx.Amount
7:              l.availBal += tx.Amount
8:          else if FPB.prepareDebit ∩ tx ≠ ∅ then               ▷ s3
9:              l.prepareDebit := l.prepareDebit ∪ tx            ▷ s4
10:             l.txnsOut := l.txsOut ∪ tx
11:             l.balance -= tx.Amount
12:         end if
13:         l.txsSync = l.txsSync ∪ tx.AcctSvcrRef
14:     end for
15:     assert txns.Balance = L.balance
16: end procedure
```

Line 2 refered by $s1$ reads bank transactions which then get processed as a credit $s2$ (incoming) payment or a debit $s3$ transaction. Lines 5 to 11 persist data onto the ledger and update the total balance. Line 13 prevents double bookings.

Processing off-ramping fiat involves two steps to reach finality. First we need to send a request to the bank-backend and only after we see the outgoing transaction in $s3$ we can update the balance on the blockchain ledger l.

Algorithm 2 Synching l with a through API

```
1: procedure PREPAREDEBIT(amt,acc)
2:     assert l.availBal ≥ amt
3:     l.prepareDebit = l.prepareDebit ∪ {creditorAccount,amount}   ▷ o4
4:     l.availBal -= amt          ▷ does not change balance yet
5:     API_{b,o}.sendPain_a(creditorAccount,amount)
6:     assert l.availBal ≥ 0                                        ▷ o1
7: end procedure
```

At $o4$ we store the prepared transactions. Similar to $s1$, we do an external call at $o1$ to the fiat backend and submit the order. Algorithm 1 will look for the transaction in the Camt.054 message and settle the transaction by updating the ledger. In case of instant ISO20022 payment notification support [10], these two steps can be merged into one short transaction.

4 Consistency and completeness

4.1 Proof of balance and transaction inclusion

The transactions of financial systems are not publicly accessible - only account holders are able to read balance and transaction data. This means the system goal of *data availability* is not supported by traditional financial systems. Financial data is classified as personal data which put data protection [22] laws in place. But with zero knowledge proofs, *HyperFridge* an achieve data availability and privacy at the same time.

Similar to blockchain ledgers, financial backends PKI infrastructure. We can use existing cryptographic protocols and secrets to "glue systems together" with zero-knowledge algorithms (circuits) to generate and later validate these proofs.

Existing banking API standards like the ISO20022 allow us to create generic solutions. Today, service providers like yapily.com

or nordeaopenbanking.com are able to connect to thousands of banks in a similar way, as we foresee it for *HyperFridge* . The ISO20022 standard [17] provides "A single standardization approach (methodology, process, repository) to be used by all financial standards initiatives" for payments, securities, trade services and more. One of its implementations is the "Electronic Banking Internet Communication Standard" (EBICS) [11], which implements standardized messages for bank account statements and gives access to typical services related to bank accounts. It has become a major element in technology strategies of banks, because central banks are enforcing and monitoring the implementation of these standards [9]. Thus ISO20022 plays an important role for the present and future international electronic payment infrastructure.

Basis of this infrastructure are permissioned APIs (of banks) which enable customers to
a) retrieve historic transaction data of a bank account and
b) to create new transactions (wire-transfers) via API.

The protocol involves key-ceremonies, uses asymmetrical cryptographic processes over the public internet safely and enforces client and server-side signatures. Data is transmitted with XML documents (or "messages"). There are other standards like PSD2 which use a more modern JSON format - but we decided to stick with ISO20022 due its adaption and maturity.

EBICS calls for multiple levels of signatures. Every message request and response carries a signature which is wrapped within a $< AuthSignature >$ XML element using the XML Signature methodology. In addition to the identification and authentication signature, electronic signatures (ES) can also be present on an EBICS payment instructions, embodied within the $< UserSignatureData >$ element.

The messages used in this paper (Camt.053 and Pain.001) are widely adopted in the financial industry (especially of Europe) for client-to-bank, bank-to-bank and bank-to-centralbank communication [7].

Information on API endpoints (e.g. $SHA256(KAC_p)$) are publicly available for many banks - see figure 4 as an example. EBICS clients which connect to bank APIs support using X.509 or passwords to verify digital signatures and keys for authentication, encryption, and electronic signatures. Examples for EBICS clients are: IBM Sterling B2B Integrator [16], elcimai.com (with Sandbox), Java-Spring based [13] or the Kotlin based "LibEuFin" from taler.net project. Libraries for parsing Camt.053 and Pain.001 files are available in languages like Rust, Go, Javascript or Java.

After the successful key ceremony between the bank and the client we have following keys available for communicating with the fiat backend, given the version H005 (EBICS 3.0 [11, Ebics 3, Appendix (see Chapter 14]) protocol:

- *X002*: for the authentication signature - KAC_p KAC_s, $SHA256(KAC_p)$ must be used by clients to validate signature.
- *E002*: for the encryption, may also use KAC_p KAC_s, but not relant for our protocol
- *A005 and A006*: for the electronic signature of the client - KES_s, KES_p

We will cryptograficly prove consistency with on-chain and off-chain data using zk-SNARKs, which offer the properties of Completeness, Soundness and Zero-knowledge [8, 3, 4]. The proof needs to span all messages seamlessly, thus all state changes of fiat backend need to be validatable on the ledger. We split the proof system in two steps:

- Step α: Prove that the full (signed) message from the fiat backend can be decrypted by the receiver $M = KAC_P(KES_s(M))$ and was processed in the right sequence (without gaps) $(M_n)_{n=0}^{\infty}$, whereas m refers to the daily statement message (Camt.053).
- Step β: Based on α prove that a transaction tx is part of a message m and all transactions of a message were processed.

4.1.1 Trusted setup

Zk-SNARKs require a trustworthy setup [5]. Hence $SNARK_\alpha$ and $SNARK_\beta$ need a secure an initial generation event of the keys needed to construct the proofs required for private transactions, as well as the verification of those proofs. This only needs to be done once. When such keys are established, a secret parameter is associated between the verification key and the keys transmitting private transactions. A potential key ceremony can be "Powers-of-Tau" [18] to minimize the attack vector to generate arbitrary claims toward the system.

4.1.2 Proving authenticity - step α

A zk-SNARK starts with proofer (operates B) that wants to prove to a verifier that he or she knows an input w such that $z = f(x,w)$, where x is a publicly known input to the function f, and x is a secret set of initial parameters. Setting up the SNARKs and using its circuit produces a public CRS (Common Reference String) s which can be used by verifiers for validation.

For each (pegging) bank account (and account holder) we need to generate a public CRS S_p using a predefined circuit. As such, the SNARK for step α is defined as

$$SNARK_\alpha := (x{:}\{SHA256(KAC_p), KES_s\}, w{:}\{M', IBAN\}) \rightarrow SHA256(M_{sliced})$$

We check encryption and signatures of the document generated by the fiat backend to ensure origin (bank) and receiver (account holder) of the bank statement.

4.1.3 Proving consistency - step β

Achieving consistency for each Message M needs to be on two levels - the bank account (overall balance) which is achieved with step α but also for each transaction (in and out) on the bank account. Hence we need to be able to compare the balance and the transactions of M with on-chain data - but data privacy prohibits us to publish M.

In EBICS M is based on *typed* XML following a standardized schemata, which allows us to safely slice the XML file into header and a remainder of transactions, symbolized by the pipe symbol "|": $M = header\,|\,(tx_0..tx_n)$. Due to the structure of M as shown in 3 (a client can have more than one bank account), we need add the $statement\ s$ to the equation to splice M:
$M = header|(statement_0(tx_{00}..tx_{n0})|statement_m(tx_{0m}..tx_{nm})$
To ensure consistency of second step β), we need to ensure that:

- A user who sent funds to the bank account needs to be able to reproduce the proof for this transaction based on his bank transaction. Therefore be able to reveal the specific part of M which a secret: $secret \rightarrow tx_m n \rightarrow statement_n \rightarrow header$ in *clear text*. The properties of step α) will proof the authenticity of M.
- Any change on-chain links publicly to tx_{mn} and hence $header$ and $statement_m$ in *clear text*. Again, the properties of Step α) will proof the authenticity of M.

For each transaction we need to create a SNARK as proof for each transaction:

Figure 3: Simplified structure of ISO20022, camt.053 message

```
CAMT.053
└─ Group Header
   └─ Statement:  There will be at least 1
      │           Statement (for each bank
      │           account).
      ├─ Balance:  Contains the Opening and
      │            Closing Balance.
      └─ Entry:  There may be multiple
         │       entries.
         └─ Details:  There may be
                      multiple entry
                      details.
```

$$SNARK_\beta := (x:\{SHA256(KAC_p), KES_s\},\ w:\{SHA256(head|statement_m|tx_{mn}))$$

We define the function

$$h = MHT(M) \rightarrow (head|statement_0|tx_{00},...,(head|statement_m|tx_{mn}))$$

which splices the daily statement document into multiple smaller pieces, each contains general data an one specific transaction.

For all snippets (each transaction of a bank account) we can validate the existence of a $SNARK_\beta$:

$$\forall tx \in m \bullet \exists SNARK_{tx_{\beta n}} \bullet (x:\{SHA256(KAC_s)\}, w:\{m\}) \rightarrow SHA256(m)$$

4.1.4 Data Availability

To reach our system goals we need to make proof data publicly available. Public data can be on-chain (smart contracts, Layer-1 Chain e.g. with Polkadot/substrate) or stored off-chain (e.g. IPFS). Validators will make sure that on- and off-chain data is consistent and can be safely used by e.g. smart contracts in a zero-trust manner. SNARKS are generated off-chain but its validators can be implemented on various protocols (e.g. etherum, polkadot, polygon), but also multiple data storage implementations are thinkable (off-chain workers, IPFS, HTTP). So one bank account can be "fridged" by one *HyperFridge* node onto several protocols without creating inconsistencies.

Data which needs to be accessible for each bank account:

- Root of the Verkle tree of all transactions is stored on-chain.
- All *encrypted* daily statements M' need to be accessible in order to check signatures and validate transaction data contained in the daily statements.
- Commitments $SNARK_\alpha$ to validate each M' and for each transaction another commitment of $SNARK_\beta$. $SNARK_\alpha$ can be on-chain, but commitments of $SNARK_\beta$ ask for being stored offline due to high transaction and data volume.

4.1.5 Atomicity

The proposed algorithm 1 does not cover timeouts, thus can not be seen as atomic in the case of off-ramping.

- Off-ramping may result in locked balances, see $\boxed{s4}$. Available balance on the ledger $l.availBal$ and the balance of the daily bank statement $txns.Balance$ will separate further on each stuck payment instruction.
- On-ramping succeeds at any point of time without timeouts or potential errors. Transactions included in daily statements (Camt.053) will be reflected accordingly.

Timeouts and exceptions should be handled one layer above (e.g. smart contracts), so that *HyperFridge* only reflects messages and state changes in both directions without adding logic to it. Only with additional logic (but building on properties of *HyperFridge*) atomic swaps can be implemented for e.g. exchanging a stablecoins with fiat using simple wire transfers.

4.1.6 Instant payments

Instant Payments will be the focus of the European Payment Council (EPC) for the next few years. Instant payments are building on top of existing SEPA payment transactions and the already established processes and implementations (SEPA: Single European Payments Area [2, 24]). Target currency is the Euro, but accounts can be in other currencies with automatic currency conversions. A key feature of instant payments is the length of time between transmission of a validated order from the originator's bank until the beneficiary's bank reports back - which should not exceed 10 seconds.

A single instant payments payment is limited to EUR 15,000 for security reasons. It will be available within the countries of the SEPA area using EBICS protocol (from EBICS 2.5). Assuming a bank supports instant payments, the off-ramping of fiat can be done in an blocking manner on the frontend for fiat/token swaps, enabling a comparable user experience to Paypal.

5 Zero trust and attack vectors

Consider following higher-level attack vectors:

Trusting the fiat backend: We assume that banks can be trusted, because their operations are highly regulated and audited; and can be compared to "proof of authority". The API of a bank uses PKI infrastructure, their public keys (certificates) are well known and used by $SNARK_\alpha$ But we can create a fake bank with a fake website and fake API. This is addressed by creating a public repository of certificates which can be managed by a community driven DAO structure, to achieve better decentralization. Validators can check the whitelists provided by the *HyperFridge* DAO.

Trusting the bank account holder: Can a holder o create $SNARK_\beta$ without having a bank account at at a specific bank? As the SNARK's public input consists of the public key of the bank and the circuit is checking signatures - so this is not possible. While creating the $SNARK_\beta$ using m , bank using signatures (see Table 1) need to be checked as well with the circuit of $SNARK_\beta$.

Withdrawal of fiat assets by the account holder: The *HyperFridge* can not prevent the account holder to withdraw funds. The system only captures current balances and transactions, and only looks back in time. Similar to blockchain-bridges [15, 25] or payment channels [21] we can collateralize the maximum amount of "open transactions" and settle payments immediately without waiting for inclusion of the transaction in Camt.053 (see 3.4).

Decrypting public Camt.054 messages (m): Storing Camt.054 messages publicly might be too risky, even if we encrypt it with a *account holder* secret keys. We split a daily statement potentially containing thousands of transactions into equally thousands of documents, which include a single transaction and the balance. In case a client secret gets

lost, only one client data gets exposed. Ideally we would see the property of "forward secrecy" - limiting a loss of key to one transaction [6]. Security on data is not considered here in further detail.

Complexity of the circiuts:

Attack vectors regarding SNARKS need to be looked in detail as well as lower-level attack vectors in case of an implementation need analyses.

6 Conclusion and outlook

We have proposed a system which allows to interact with fiat backends in a trustless manner based on zero-knowledge proofs. Without compromising on privacy, anyone can validate fiat balances, prove the inclusion of (own) transactions and sending funds through third party bank accounts; both taking finality of fiat transactions into account. The system can operate with multiple ledgers protocols or even without a ledger. It uses battle-tested and widely adopted standards of the financial industry with makes it easy to use *HyperFridge* nodes in many countries. Some examples of use cases:

- Currenlty "proof of solvency" is widely discussed - *HyperFridge* can offer decentralized infrastructure to prove liabilities of stablecoins (pegging ratio) and therefore provide a transparency layer even for whitelabeled or private stablecoins.
- Using *HyperFridge* , smart contracts can operate directly on fiat accounts, eliminating the need of converting into tokens. Hence real world fiat use cases (e.g. trade finance, real estate) can benefit from smart contracts by adding logic to a bank account in a trustless and transparent manner - a pattern which currenlty can only be explored in the DeFi world of blockchain.
- Identity validation is intrinsic to banking networks and thus to *HyperFridge* as well: An account holder is always authenticated by the bank, and a global regime (FATF - Financial Action Task Force) is auditing global standards on the maturity of identity and governance processes. Thus *HyperFridge* can act as a strong source of identity for individuals and organizations.
- We can extend the system to other banking protocols like the FIX protocol (Financial Information Exchange), which has became the standard protocol for pre-trade communications and trade execution for traditional assets. Thinking of the FTX collapse - if assets would have mirrored with *HyperFridge* , then it would have been transparent from the beginning how your funds where used and if your funds are still with FTX. Key learning here is, that most information we need is already in some systems, which likely can be accessed via an API.

In this document we refer to ISO20022 which is widely used within the "Single European Payments Area" (SEPA) where currently 36 countries are part of. We are confident that *HyperFridge* system goals can also be reached with other payment standards which make use of PKI infrastructure. *HyperFridge* would then be able to provide a single protocoll and trustless access to banking infrastructure worldwide.

We believe that the *HyperFridge* system model has the potential to reshape the solution space of both traditional and crypto finance industry.

7 Appendix

Name	Description
A004	EBICS A004 electronic signature. RSA key 1024 bit for a ISO9796-2 DINSIG signature. Hashalgorithm is RIPEMD-160.
A005	EBICS A005 electronic signature. RSA 1536 bits to 4096 bits for a PKCS #1 signature. Hashalgorithm is SHA-256.
A006	EBICS A006 electronic signature. RSA 1536 bits to 4096 bits for a PKCS PSS signature. Hashalgorithm is SHA-256.
E001	EBICS E001 encryption (2-Key-Tripple-DES). RSA 1024 bits to 16384 bits and PKCS #1 padding.
E002	EBICS E002 encryption (AES-128). RSA 1024 bits to 16384 bits and PKCS #1 padding.
X001	EBICS X001 authentication signature. RSA 1024 bits to 16384 bits for a PKCS #1 signature.
X002	EBICS X002 authentication signature. RSA 1024 bits to 16384 bits for a PKCS #1 signature. Hashalgorithm is SHA-256.

Table 1: Signatures and Encryption used by ISO20022

Selected relations to ISO20022 messages			
Name	XML Field	Example	used as
Cash Management: $B.getStmts() \rightarrow$ camt.053.001.10			
Amount	$<Amt>$	USD 5000	$tx.Amount$
Credit Debit Indicator	$<CdtDbtInd>$	CRDT	$tx.CredDeb$
Account	$<Acct>$	AT48..864	$tx.Account$
Status	$<Sts>$	BOOK	$tx.Status$
Account Servicer Reference	$<AcctSvcrRef>$	AASS-ACCR-01	$tx.AcctSvcrRef$
Payments Initiation: $B.pay(pain.001.001.11)$			
Debtor Account	$<DbtrAcct>$	AT48..864	pegging account
EndToEndId	$<EndToEndId>$	ABC	needed by $\boxed{s1}$
Creditor	$<Cdtr>$	DE12..567	$B.pay(acc,amt)$
Instructed Amount	$<InstAmt>$	USD 50000	$B.pay(acc,amt)$
Purpose	$<Purp>$	0x34	e.g. etherum transaction hash

Table 2: Using the standarized messages of financial protocols

Figure 4: Example of ISO20022 access information, [1]

Parameter for Connecting, Order Types and File Formats EBICS Bank CIC (Switzerland) Ltd.

Parameters for connecting to EBICS

Host ID	BANKCICCH
URL	https://ebics.cic.ch/EBICS-Server/ebics.aspx
Port	443
EBICS version	EBICS 2.4 (H003) and EBICS 2.5 (H004)
Accepted keys	Encryption key: E002, Authentication key: X002 Electronic signature: A005 and A006

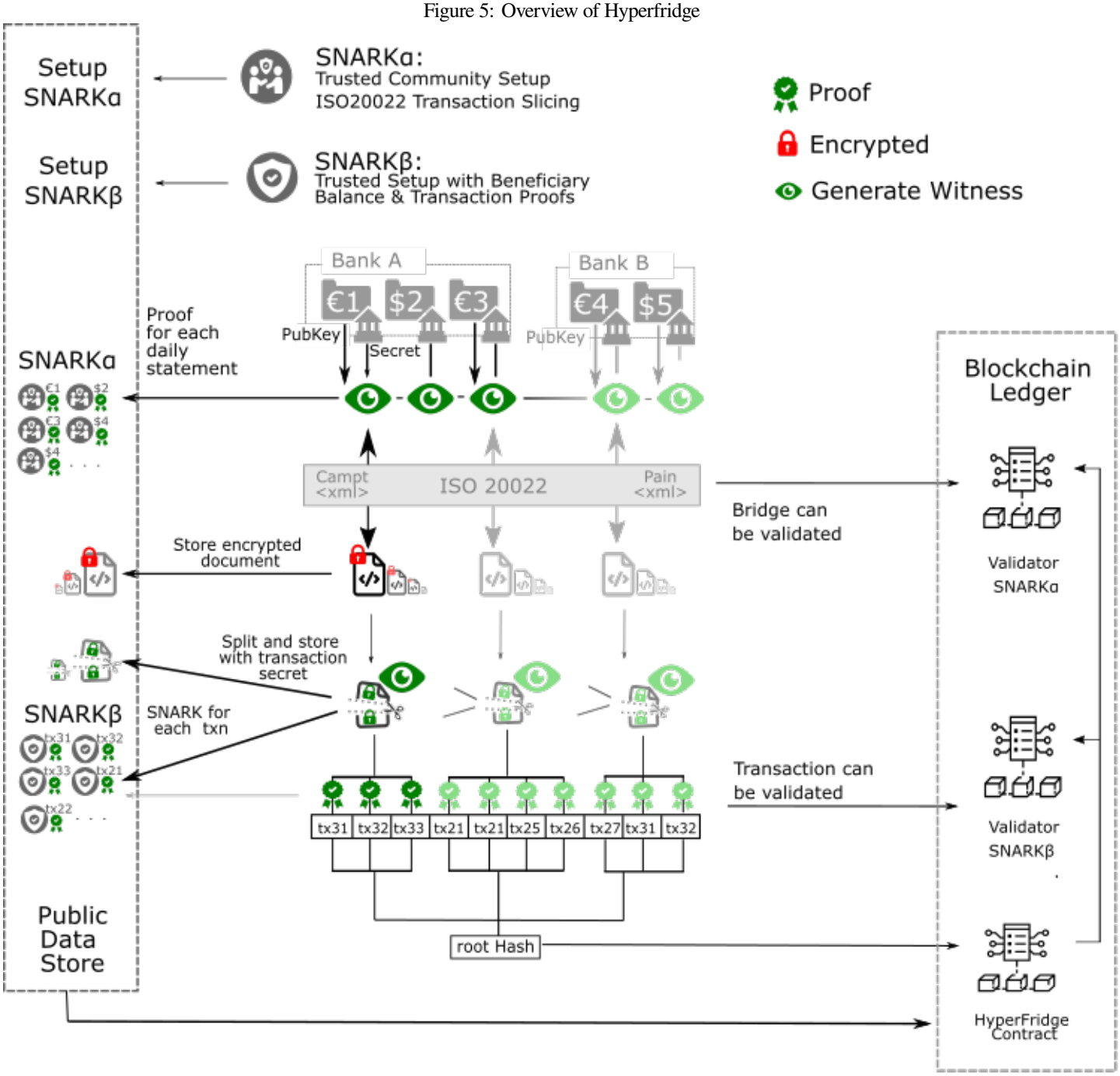
Hash values of Bank CIC (Switzerland) Ltd.

EBICS encryption key
ee 8f 5b 40 3c 9a e9 b1 cc 9b f7 79 21 cb 7f 36 ec bc d8 54 63 ec 55 9a 7b b2 4f 72 a5 ee 07 03

EBICS authentication key
57 d0 68 d0 0a 95 cf 1e d0 8b c3 af 06 98 92 1b 47 21 62 e3 af ee 43 2d cd 3d ff 6d 8e 4d 07 5e

To establish a secure connection with the Bank, the EBICS User utilises his EBICS keys, as well as the Bank's keys, whose accuracy he must verify before using EBICS. The Bank's current keys can be retrieved at any time from its website: www.cic.ch/ebics. The Bank's keys enable the EBICS User to verify whether he is connected to the Bank. The EBICS User is exclusively responsible for establishing and verifying the connection with the Bank.

Figure 5: Overview of Hyperfridge



References

[1] CIC Bank. Parameter for connecting, order types and file formats ebics bank cic (switzerland) ltd. https://www.cic.ch/dam/jcr:f47b77c1-db6a-4850-9ca6-c3f364907820/connection-parameters-ebics-bank-cic.pdf.

[2] European Central Bank. National authorities in sepa countries that issue bank identifiers used in ibans. *europa.eu*, 2014.

[3] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again, 2012.

[4] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. *Journal of Cryptology*, 35, 2013.

[5] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *IACR Cryptol. ePrint Arch.*, 2017:1050, 2017.

[6] Colin Boyd and Kai Gellert. A modern view on forward security. Cryptology ePrint Archive, Paper 2019/1362, 2019. https://eprint.iacr.org/2019/1362x1.

[7] Deutsche Bundesbank. Ebics procedural rules, effective from 21 november 2021.

[8] Matteo Campanelli and Hamidreza Khoshakhlagh. Succinct publicly-certifiable proofs - or, can a blockchain verify a designated-verifier proof? In *INDOCRYPT*, 2021.

[9] EBA Clearing. Iso 20022 migration. https://www.ebaclearing.eu/services/euro1/iso-20022-migration/.

[10] EBA Clearing. Target2/t2s consolidation. https://www.bundesbank.de/en/tasks/payment-systems/target2-t2s-consolidation/target2-t2s-consolidation-723780.

[11] EBICS. Ebics - electronic banking internet communication standard. https://www.ebics.org/en/home.

[12] Switzerland element36 AG. Cash36 - private fully pegged stablecoins for fiat on- off-ramp. https://www.element36.io.

[13] Walter Strametz et al. Ebics-java-service. https://github.com/element36-io/ebics-java-service.

[14] Web3 Foundation. Polkadot website. https://polkadot.network/.

[15] Thomas Hardjono. Blockchain gateways, bridges and delegated hash-locks, 2021.

[16] IBM. Ibm sterling b2b integrator. https://www.ibm.com/docs/nl/b2b-integrator/5.2?topic=eckf-managing-certificates-keys-users.

[17] ISO. Iso 20022 - universal financial industry message scheme. https://www.iso20022.org/.

[18] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. Cryptology ePrint Archive, Paper 2022/1592, 2022.

[19] Advisory Group on Market Infrastructures for Securities and Collateral. 10th t2s harmonisation progress report. https://www.ecb.europa.eu/paym/intro/publications/pdf/ecb.tenth_t2sharmonisationprogressreport.en.pdf.

[20] European Parliament. Directive 98/26/ec. https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:31998L0026.

[21] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2016.

[22] European Parliamentary Research Service. Blockchain and the general data protection regulation. https://www.europarl.europa.eu/RegData/etudes/STUD/2019/634445/EPRS_STU(2019)634445_EN.pdf.

[23] Walter Strametz. Fiat bridge grant. https://github.com/element36-io/Open-Grants-Program/blob/master/applications/FIAT-on-off-ramp.m.

[24] Siegfried Trapp. Faqs vollständige umstellung auf sepa (einheitlicher euro-zahlungsverkehrsraum). 2010.

[25] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William J. Knottenbelt. Xclaim: Trustless, interoperable cryptocurrency-backed assets. Cryptology ePrint Archive, Paper 2018/643, 2018.