BitML^x– Cross-chain Smart Contracts for Bitcoin-style Cryptocurrencies (Work in Progress)

Abstract—**The limited scripting capabilities in Bitcoin-like cryptocurrencies have forced implementations of smart contracts as multi-party cryptographic protocols. To streamline this process, the BitML language allows for defining simple smart contracts and automatically translates them into protocols over transactions in the respective currency. However, BitML is limited to contracts operating upon the same cryptocurrency whereas many interesting financial applications involve assets on different blockchains, inducing more complicated cryptographic protocols for enforcing synchronous execution across these systems. In this work, we introduce BitML^x, an extension of BitML that provides a high-level programming language to implement smart contracts executing synchronously on any two Bitcoin-like cryptocurrencies. We provide a compiler from BitML^x to two BitML contracts and formally prove that participants executing the latter contracts end up at least as good as in the corresponding execution of the former BitML^x contract.**

Index Terms—**smart contracts, Bitcoin, cross-chain, compiler**

I. Introduction

Smart contracts are programs controlling cryptocurrency assets and serve as trustless implementations of many financial applications, such as escrow services or lotteries. While some cryptocurrencies (like Ethereum) support a (quasi) Turing-complete language, Bitcoin (and similar cryptocurrencies) support only very limited scripting capabilities to express conditions on how individual coins can be spent. Despite that, there exist many examples [2]–[5], [7]–[11] of Bitcoin-compatible smart contracts designed as multi-party cryptographic protocols where conditions to spend the participants' coins are carefully intertwined to realize the logic of the contract. A more principled way for designing such protocols is provided by BitML [1], a high-level programming language for smart contracts that can be compiled to protocols over Bitcoin transactions with formal soundness guarantees.

In practice, many interesting financial applications involve assets in different cryptocurrencies. However, synchronizing the execution of smart contracts across several blockchains is notoriously hard since one cannot rely on the synchronicity of the consensus mechanism underlying the individual cryptocurrencies. Instead, the protocol parties need to synchronize their actions by cryptographic means, resulting in inherent fairness issues known from secure multi-party computation [6]. These limitations need to be overcome with carefully crafted financial incentives, a factor that substantially complicates the design of secure contracts.

To address this challenge, we introduce BitML^x, an extension of BitML, which models contracts executing synchronously on any two cryptocurrencies that support Bitcoin-like scripting. We give a formal semantics for BitML^x and provide a translation to concurrently executing BitML contracts. We prove the translation correct, showing that honest users interacting with the compiled BitML contracts can always enforce an execution that ensures an outcome as good as the corresponding execution of the original BitML^x contract.

II. Background

A BitML contract governs the deposits of the contract parties according to the rules of a simple process calculus. In the following, we will introduce the main components of this process calculus using examples.

Consider a scenario where a user A holding 10 bitcoins (₿) and a user B holding 10 dogecoins (Ɖ) wish to exchange their assets. To do so, they deposit their assets into BitML smart contracts $\{A : !10₿\}\mathit{Swap}_{₿}$ on Bitcoin and $\{B : !10Ɖ\}\mathit{Swap}_{Ɖ}$ on Dogecoin. The preconditions $\{A : !10₿\}$ and $\{B : !10Ɖ\}$ denote the deposits made by the parties and the codes $\mathit{Swap}_{₿}$ and $\mathit{Swap}_{Ɖ}$ describe the contract logic. A first attempt at implementing the contract codes could look as follows:

$$\begin{aligned}
\mathit{Refund}_{₿} &= \texttt{after } t : \texttt{ withdraw } A\\
\mathit{Swap}_{₿} &= A : \texttt{ withdraw } B + \mathit{Refund}_{₿}\\
\mathit{Refund}_{Ɖ} &= \texttt{after } t : \texttt{ withdraw } B\\
\mathit{Swap}_{Ɖ} &= B : \texttt{ withdraw } A + \mathit{Refund}_{Ɖ}
\end{aligned}$$

The definition of $\mathit{Swap}_{₿}$ indicates a choice ($+$) between two actions: either (i) upon authorization from A, B can withdraw all assets in the contract (here $10₿$) or (ii) after time t, A can withdraw the contract assets. The contract $\mathit{Swap}_{Ɖ}$ is defined symmetrically. While these contracts allow parties A and B to safely deposit their assets on the corresponding blockchain and to retrieve them back in case the other party does not do the same, they are still left with a coordination problem: Whoever authorizes the transfer of their assets first has no guarantee that the other party will do the same and not just wait till time t to also claim back their own assets.

To solve this problem, the contract could use a trusted intermediary C who will authorize both transfers simultaneously, for a small fee. For example, we could replace $A : \texttt{ withdraw } B$ with the following $\mathit{Escrow}_{₿}$ contract (and analogously for the Dogecoin case):

$$\begin{aligned}
\mathit{Escrow}_{₿} = C : \texttt{ split}(&9₿ \rightarrow \texttt{withdraw } B,\\
&1₿ \rightarrow \texttt{withdraw } C)
\end{aligned}$$

$\mathit{Escrow}_{₿}$ encodes that with C's authorization, the funds are split into two independent contracts, one where B gets $9₿$, and one where C gets 1₿. A rational C will synchronize the execution of $\mathit{Swap}_{₿}$ and $\mathit{Swap}_{Ð}$ and even if C is offline, A and B can retrieve back their assets.

This security argument, however, relies on the existence of a rational third party C and the payment of a fee. In the following, we show how to achieve the synchronous execution of BitML-style smart contracts across different blockchains without the need for fees and a synchronizing party.

III. Solution Overview

We introduce BitML^x, a language for writing smart contracts that simultaneously govern assets in two Bitcoin-like cryptocurrencies. BitML^x closely resembles BitML and comes with an operational semantics for the ideal synchronous contract execution. To realize that, we translate BitML^x contracts into two BitML contracts to be executed in parallel on the respective blockchains. Finally, we prove a correctness statement relating the concurrent execution of the compiled contracts with the ideal execution of the original BitML^x contract.

The BitML^x language. We overview the BitML^x features with the asset swap example. Participants deposit their coins into a contract $\{A :!(10₿, 0Ð) \mid B :!(0₿, 10Ð)\}\mathit{Swap}^x$, where

$$\mathit{Exchange} = \texttt{split}((0₿, 10Ð) \rightarrow \texttt{withdraw}\ A, \quad (10₿, 0Ð) \rightarrow \texttt{withdraw}\ B)$$

$$\mathit{Refund} = \texttt{split}((10₿, 0Ð) \rightarrow \texttt{withdraw}\ A, \quad (0₿, 10Ð) \rightarrow \texttt{withdraw}\ B)$$

$$\mathit{Swap}^x = \mathit{Exchange} +\!> \mathit{Refund}$$

Similar to BitML, the `split` primitive splits the contract into two independent instances, each with its own funds. The key difference being that, in BitML^x, deposits are extended to tuples on both currencies, which allows us to express that, simultaneously, in one branch one participant takes all bitcoins while on the other, the other participant takes all dogecoins.

On the top level, the Swap^x contract exposes a priority choice (represented by the $+\!>$ operator) indicating that the execution of the $\mathit{Exchange}$ contract has priority over the Refund contract. BitML^x features priority choices (as opposed to a normal choice operator) to ensure a predictable execution behavior of the contract, which is a prerequisite for a synchronous execution across blockchains. Participants in a BitML^x contract are additionally required to deposit extra funds that will not intervene into the contract logic, but function as collateral to secure the synchronization mechanism. The necessary collateral can be computed from the contract's deposits and number of participants.

Compilation. BitML^x contracts are compiled to a pair of BitML contracts in the respective target blockchains, and funded with the deposits and the collateral from each participant. The control flow of the compiled contracts is tied by a mechanism of timed commitments and punishments to encourage that any time a participant takes an action on one blockchain, they replicate it on the other. Whenever progress is not replicated, the participant responsible for the asymmetrical

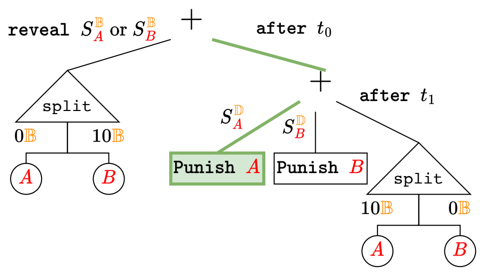

Fig. 1: Bitcoin compilation of Swap^x.

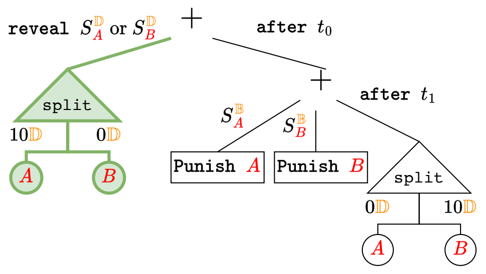

Fig. 2: Dogecoin compilation of Swap^x.

behavior is forced to pay compensation to potentially harmed participants by splitting their collateral among them.

In Figs. 1 and 2, we depict the BitML contracts resulting from the compilation of Swap^x and highlight the case (in green) where A tries to go for the exchange on the Dogecoin blockchain but, also tries to get a refund on the Bitcoin blockchain. To do so, A needs to reveal their own special secret $S_A^{Ð}$ that they committed to before starting the execution. Revealing this secret is a condition for A to take the left side of the Dogecoin contract and serves as proof of A making that step. In the Bitcoin contract (Fig. 1), after time t_0, B can use this same secret to punish A (indicated by `Punish` A). To ensure that B has sufficient time for doing so, the Refund contract will only be enabled after $t_1 > t_0$. During the punishment, B is rewarded the whole contract balance (10₿) ensuring that the asynchronous execution is as rewarding for B as the synchronous one. If the contract would involve more users, A's collateral would be used to pay them a corresponding compensation (covering the whole contract balance). In synchronous executions, such collateral is returned to the owner.

Correctness. To establish the correctness of the compilation, we show the following statement (here informal):

Theorem (Compiler correctness, informal)**.** *Each strategy of an honest user A on a BitML^x contract C translates into a strategy on the concurrently executing compiled BitML contracts $C_{₿} \mid C_{Ð}$ that allows A to extract at least as many assets from $C_{₿} \mid C_{Ð}$ as from C with the original strategy.*

Intuitively, this theorem states that users interacting with the concurrently executing BitML contracts resulting from the compilation can always achieve results at least as beneficial as the ones resulting from the interaction with the original BitML^x contract according to the synchronous semantics.

References

[1] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. A formal model of bitcoin transactions. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*, pages 541–560. Springer, 2018.

[2] Lukas Aumayr, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Kristina Hostáková, Matteo Maffei, Pedro Moreno-Sanchez, and Siavash Riahi. Generalized channels from limited blockchain scripts and adaptor signatures. In *27th International Conference on the Theory and Application of Cryptology and Information Security*, 2021.

[3] Lukas Aumayr, Matteo Maffei, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. Bitcoin-compatible virtual channels. In *42nd IEEE Symposium on Security and Privacy, SP 2021*.

[4] Lukas Aumayr, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. Breaking and fixing virtual channels: Domino attack and donner. In *Annual Network and Distributed System Security Symposium*, 2023.

[5] Lukas Aumayr, Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, Pedro Moreno-Sanchez, and Matteo Maffei. Sleepy channels: Bi-directional payment channels without watchtowers. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security,*, pages 179–192.

[6] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 364–369, 1986.

[7] Oguzhan Ersoy, Pedro Moreno-Sanchez, and Stefanie Roos. Get me out of this payment! bailout: An htlc re-routing protocol. Cryptology ePrint Archive, Paper 2022/958, 2022. https://eprint.iacr.org/2022/958.

[8] Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri Aravinda Krishnan Thyagarajan. Foundations of coin mixing services. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1259–1273.

[9] Philipp Hoenisch, Subhra Mazumdar, Pedro Moreno-Sanchez, and Sushmita Ruj. Lightswap: An atomic swap does not require timeouts at both blockchains. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2022 International Workshops, DPM 2022 and CBT 2022,*.

[10] Varun Madathil, Sri Aravinda Krishnan Thyagarajan, Dimitrios Vasilopoulos, Lloyd Fournier, Giulio Malavolta, and Pedro Moreno-Sanchez. Cryptographic oracle-based conditional payments. In *Annual Network and Distributed System Security Symposium*, 2023.

[11] Sri Aravinda Krishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps: Secure exchange of coins across all blockchains. In *43rd IEEE Symposium on Security and Privacy, SP 2022*.