Modeling Trust Relationships in Blockchain Applications: The Case of Reconfigurable Systems-on-Chip

Maxime Mere, Frederic Jouault, Loic Pallardy, Richard Perdriau

Modeling Trust Relationships in Blockchain Applications: The Case of Reconfigurable Systems-on-Chip

SUBMISSION DATE / POSTED DATE

16-09-2022 / 06-10-2022

CITATION

Méré, Maxime; Jouault, Frédéric; Perdriau, Richard; Pallardy, Loïc (2022): Modeling Trust Relationships in Blockchain Applications: The Case of Reconfigurable Systems-on-Chip. TechRxiv. Preprint. https://doi.org/10.36227/techrxiv.21135706.v2

DOI

10.36227/techrxiv.21135706.v2

Modeling Trust Relationships in Blockchain Applications: The Case of Reconfigurable Systems-on-Chip

Maxime Méré[1,2], Frédéric Jouault[3], Loïc Pallardy[2], and Richard Perdriau[4,3]
[1]Univ Rennes, INSA Rennes, CNRS, IETR – UMR 6164, F-35000 Rennes
[2]STMicroelectronics, Le Mans, France
[3]ESEO-Tech, F-49100 Angers, France
[4]CNRS, IETR – UMR 6164, F-35000 Rennes, France
maxime.mere@st.com, frederic.jouault@eseo.fr, loic.pallardy@st.com, richard.perdriau@eseo.fr

***Abstract*—Nowadays, System-on-Chip (SoC) components are found everywhere in all kinds of smart devices. Each System-on-Chip contains many different blocks that provide specific functionalities, such as WiFi or Bluetooth connectivity. Whereas integrating each such block in a SoC typically requires paying some royalties, not all blocks are necessary for all applications, or throughout a device's lifecycle. Moreover, it is not possible to manufacture a specific SoC for each application. Significant advantages are therefore expected to be gained by enabling trustworthy remote SoC reconfiguration throughout their life cycles. A few approaches attempting to address this challenge have been proposed in the literature. They are typically based on Blockchain technology in order to support decentralization without relinquishing trust. Reviewing these approaches lead us to identify a potential flaw in the proposed protocols. Indeed, a SoC should be able to trust Blockchain information that it is given, without requiring any centralization. In order to validate our suspicions, we propose in this paper to use Verifpal: a cryptographic protocol verification tool that works from textual protocol models. We use it in a slightly unorthodox way in order to model the trust relationships in one of the approaches from the literature, and to verify it. The results show that, under some assumptions, a flaw is indeed present. We propose and model several possible fixes, and present their respective limitations.**

Keywords*—*Blockchain; Trust Relationships; Re-configurable Systems-on-Chip

I. Introduction

The Internet of things comprises many applications. One of these applications is the possibility to reconfigure intellectual property (IP) blocks found in hardware components, and in particular in Systems-on-Chip (SoCs). This type of application is not widespread yet because component reconfiguration requires a high level of security that is difficult to achieve. The main reason is that the components to reconfigure are typically in an untrusted environment. The literature on approaches that offer these possibilities is, according to the authors' knowledge, still thin [1]. The proposed approaches use Blockchain technology to obtain security guarantees [2], [3]. Blockchain is a technology originally designed to offer a decentralized currency system capable to work without trusted third parties [4]. The possibilities offered by Blockchain technology have been extended to a wider set of applications with the smart contract concept. These are programs that automatically run on a Blockchain network, and whose results are immutable [5]. The purpose of Blockchain in SoC reconfiguration applications is to avoid third parties and to decentralize trust. Setting up the trust link between a chip and Blockchain data is not a trivial task. Moreover, the published protocol descriptions lead us to believe that trust flaws can appear when a component interacts with the Blockchain network. It is because of this potential design flaw that we propose to model and verify an approach from the literature, along with potential fixes. To this end, we use Verifpal, which is a tool designed to be relatively easy to use, and which can be used to model and verify cryptographic protocols [6]. All of the Verifpal models created for this work, as well as additional figures, are available on GitHub[1]. The rest of the paper is organized as follows. Section II presents the context of this work, and more especially what is modeled. Related work is presented in section III. The modeling approach and an overview of Verifpal are given in section IV. Section V, presents the model, the verification results, and describes possible fixes. Notes on potential threats to the validity of the results are presented in section VI. Finally, section VII concludes and presents some perspectives as well as some future work.

II. Context

Allowing the unconstrained reconfiguration of a component makes it possible to offer many functionalities either during production or in the finished products. Thus, it is envisioned to offer "à la carte" components to reduce the number of SoC references, to allow dynamic royalty payments, or even to enable the rental of hardware features. We call *life cycle management system* (LCMS) [1] the sub-system which, when integratred into a SoC, makes it capable of being reconfigured throughout its life cycle. In the scope of this paper, we focus on the protocol proposed by Islam et al [2]. However, a similar problem seems to be present in other approaches, such as [3], because they work on the same principles. Figure 1 represents a simplified version of the protocol as a sequence diagram. The different participants in this diagram are the following:

[1]https://github.com/meremST/trustability-model

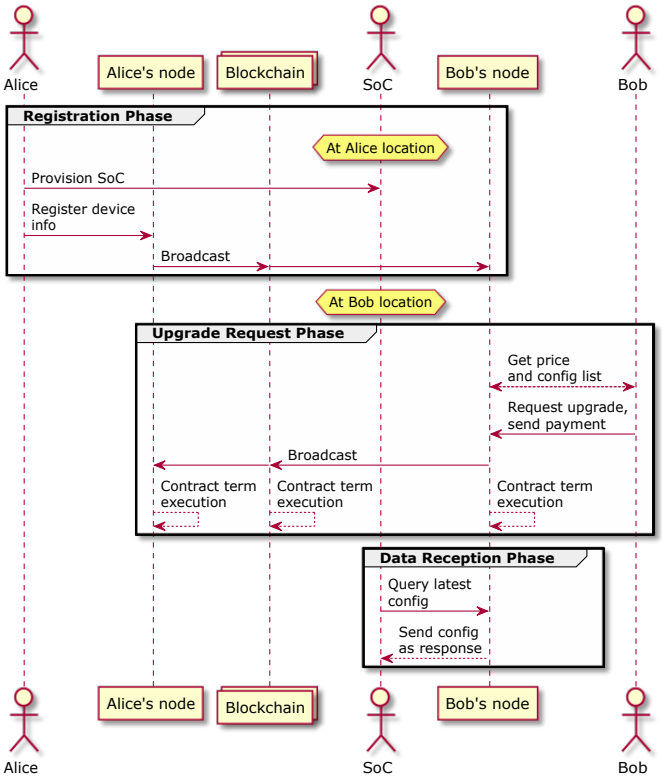

Figure 1. Sequence diagram of state of the art Blockchain-based SoC reconfiguration protocol

- *Alice* is the SoC manufacturer.
- *Bob* is the buyer of the SoC.
- *SoC* is the reconfigurable component.
- Nodes *Alice's node* and *Bob's node* are Blockchain nodes respectively owned by Alice and Bob. Since Alice and Bob do not, in general, trust each other, they do not necessarily trust the other party's node.
- *Blockchain*: corresponds to all the other nodes that constitute the Blockchain. The presence of this Blockchain participant makes it possible to model Blockchain-generated consensus. By definition, the data it stores is immutable.

The protocol is composed of three separate phases:

- *The registration phase* corresponds to the moment when Alice (the manufacturer) loads secrets in or retrieves secrets from the SoC. These secrets can be very diverse, such as identification means, the price and list of possible configurations, or generated cryptographic elements. With this information, Alice is then capable of creating a smart contract in the Blockchain that aims at allowing future owners to purchase configurations.
- *The modification or upgrade request phase* is the moment when Bob (the SoC owner) will retrieve available configurations, and will use the smart contract to buy one of them. The payment is then validated by the Blockchain, and the contract status changes to indicate that the user now has access to a specific configuration.
- *The data reception phase* is when the SoC retrieves its enabled configuration from the Blockchain. The Blockchain being by definition immutable, it is not possible to forge this data.

The potential flaw of this protocol happens at the moment of data reception by the SoC. If Bob could trick the SoC into erroneously believing that he has actually paid for a given configuration, then he would be able to access it without paying. However, directly checking Blockchain information is a time-consuming action that requires a large amount of computing power [7], which is not actually available on this type of component. The SoC must therefore trust a specific node to give it reliable information.

III. Related Work

The notion of trust is complex and can be interpreted in several ways. Ruan and Durresi [8] summarize in their survey how trust can be modeled in online social communities. Their work is general enough to be applicable to devices and covers a substantial part of the literature on this subject. The authors conclude that each representation of trust depends on the definition of trust on which it is based. Moreover, they present different trust management scheme models. Compared to this work, our needs in trust modeling are much simpler because we only need to determine if a trust relationship is established between two actors following a protocol. In the context of research on the use of Blockchain technology in electronic objects, Miraz [9] proposed a summary of the benefits as well as of the challenges that this type of technology presents. Solutions have been proposed to address interactions between a Blockchain and the external environment. Caldarelli [10] provides an overview of the research on oracles, which allow the use of off-chain data in smart contracts. To the best of our knowledge, oracles are not capable of reliably pushing data out of the Blockchain, as required in our case. Other studies have focused on modeling blockchain-based systems. Sukhwani et al. [11] proposed a performance model of Hyperledger fabric, a permissioned Blockchain network, to evaluate its performance. Rocha and Ducassse [12] have worked on how to model software based on smart contracts using UML. Like Verifpal, there are several tools that can model protocols in order to verify them. Among the most popular are ProVerif and Tamarin. ProVerif is a tool by Bruno Blanchet [13], which is able to automatically translate a protocol description into Horn clauses, and to determine whether the desired security properties hold by resolution on these clauses. The Tamarin prover, proposed by Meier et al [14], is a tool that supports the automated, unbounded, symbolic analysis of security protocols. In the case of our work, Verifpal was chosen because it is comparatively easier to use but still powerful enough to be able to support the models we need to create.

IV. Approach Overview

In order to verify if the trust flaw can be identified, we propose to model the protocol, and verify it. To do this, it is necessary to model trust by using a tool able to perform verification.

A. Modeling Trust

There are several ways to formally model the protocol from Figure 1. It should be possible to precisely model the behavior of the Blockchain, and notably the different nodes that compose it, but such a model could be hard to check. Alternatively, it is also possible to consider a more abstract model that focuses on the trust relationships between the different actors. It is possible to model trust relationships with the following principles.

- The set of *honest* nodes that compose the Blockchain network can be regrouped into a single actor.
- The nodes used by the different actors must be modeled separately because any actor could subvert their own nodes. This means that a given actor will trust their nodes but will not blindly trust the other actor's nodes and *vice versa*.
- Data contained in the Blockchain is public. The Blockchain can therefore send and receive messages, and it has the ability to perform operations on its data, but it is not able to store a private key. Consequently, the Blockchain cannot directly encrypt or sign data.
- Any data stored in the Blockchain is considered unforgeable. It is important to point out that although data inside the Blockchain is immutable, in order to read it with great confidence, it is necessary to be able to cryptographically verify it. The only actors able to directly read the Blockchain are the nodes.

By applying these rules and modeling the message exchanges that describe the protocol, it is possible to create a verifiable model on which a verification algorithm will be able to find a counterexample if a flaw exists.

B. Verifpal

Verifpal[2] is a textual modeling and verification framework for cryptographic protocols [6]. It is designed to make it relatively easy to obtain a working model and to verify properties. Verifpal offers formal verification for confidentiality, authentication, freshness, and unlinkability properties. It is relatively close to a textual sequence diagram specification. It uses pre-defined cryptographic primitives to describe all protocol cryptographic operations. This has the advantage of preventing the use of ill-defined primitives. To check the security of a protocol, Verifpal is based on the Dolev-Yao model [15]. Verification uses an algorithm that performs transformations on protocol messages. These transformations come from a set of mutations that an attacker would be able to perform from the information that can be gathered by message interception, or that can be reconstructed. Verifpal then checks if it has found a contradiction to one of the queries (i.e., properties) specified in the model. Verifpal allows to relatively easily and efficiently model a given protocol. However, because of its relative youth, its performance has not been significantly evaluated yet. The choice to use this tool was motivated by its simplicity of use and the speed with which one can obtain a working result. A Verifpal model is divided into three parts.

- First comes the definition of attacker behavior.
- Then comes the protocol model consisting of the principals (i.e., the participants) and the message exchanges.
- Finally, the query section allows to formulate the properties to verify.

Attacker behavior. The attacker can be set as passive or active. A passive attacker can access all exchanged messages, and will try to recover secrets from these messages. An active attacker also has the possibility to inject messages, which corresponds to the Dolev-Yao attacker. Attacker declaration is performed with `attacker[passive]` or `attacker[active]` at the beginning of the model.

Protocol modeling. This is the core part of a Verifpal model. The description of a protocol consists of two elements, namely the declaration of *principals*, and the transmission of *messages*. Each participants and its actions are declared with the `principal` keyword, followed by the actor's name. The actions on the data performed by the actor can be described inside brackets after its name. It is possible to use `principal` whenever it is necessary for an actor to process data. Possible data processing can be either the generation of constants, or the use of cryptographic primitives. The possibility to declare constants allows the representation of data in general. It should be noted that Verifpal does not allow mutable variable declaration, thus once declared, variables cannot be reassigned. There are two ways to declare a constant.

- A principal can acquire the knowledge of any constant via the `knows` statement. If the constant does not already exist, this declares it. `knows` is associated with a `private` or `public` qualifier to determine if the constant is supposed to be considered as known by everyone else.
- The `generate` function allows a principal to declare a value generated on the fly. This corresponds to values such as temporary session keys or random numbers that are replaced at each iteration of the protocol.

A participant can later reveal a constant to an attacker via the *leak* keyword in order to simulate data theft. Regarding cryptographic primitives, they correspond to the basic functions that allow to describe cryptographic protocols. These functions are predefined in Verifpal, and are considered as "perfect". This means they represent the theoretical operation of a cryptographic concept, and not any implementation with its possible limitations. There are many tools among these primitives to describe as many different protocols as possible. In this article, only those used in the models are presented, that is: public and secret key generation, signatures and hash functions. Verifpal allows the use of equations, which are used to represent mathematical properties that make it possible to generate public keys from private keys, or to establish shared secrets. These equations are presented in the following form for the generation of public keys: `publicKey=G^privateKey`. Each key pair allows to perform asymmetric encryption or to sign messages. Signatures are carried out via primitive

[2]In this work, we used version 0.27.

`signedm=SIGN(k, m)`, with `k` a private key, `m` the message to sign, and `signedm` the signed message. It is possible to check this signature with primitive `_=SIGNVERIF(Gˆk, m, signedm)?`, which will check that the signed message `signedm` corresponds to `m` signed with private key `k`. Other available primitives are one-way functions. They are performed with primitive `h=HASH(a,b,...)`. Result `h` is easy to obtain from the given input but this input is impossible to recover from `h`. To conclude about primitives, there is a function that checks the equality of two values. It is presented in the form `_=ASSERT(c1, c2)?` where `c1` and `c2` are the values to compare.

Once the principals have been declared, and the constants generated and processed with primitives, the exchanges between the participants are expressed through messages. Messages are in the form `A->B: m,[k]`. In this example `A` is sending a message to `B` that contains values `m` and `k`. These can be constants or the result of the combination of any available primitives previously calculated in a sender principal. It is through messages that an attacker will be able to break the protocol, when it is flawed. In the message example presented above, constant `k` is written between brackets. That means that it is guarded: an active attacker can still read it, but not tamper with it. Guarded constants are useful to simplify models, and to consider that the data in question is already pre-authenticated. Finally, Verifpal offers a system of phases, which makes it possible to compartmentalize the protocol. This can be used to avoid state space explosion.

Query section. The last part of a Verifpal model is the query section. Verifpal models are verified on different properties that are defined as queries. The queries are expressed inside the brackets of the `queries[]` statement. There are four types of queries that can be checked.

- *Confidentiality* queries are the simplest. This is equivalent to asking: "can the attacker get a certain constant?" It is expressed in the form `confidentiality? m`, with `m` a secret constant.
- *Authentication* queries are used to verify that an attacker cannot modify a message without the receiver noticing. It is formulated as follows: `authentication? Alice->Bob: m`. Here, it is the constant `m` sent to Bob by Alice that is checked.
- *Unlinkability* queries are related to voting protocols. They are used to verify that an attacker is not able to distinguish two constants during protocol execution. They are expressed as `unlinkability? m1, m2`, with `m1` and `m2` the two constants which must be indistinguishable.
- *Freshness* queries are used to ensure that it is not possible to perform replay attacks. Their use is beyond the scope of this paper.

C. Modeling Trust Relationships with Verifpal

As previously explained, Verifpal is a tool capable of modeling cryptographic protocols in order to check specific properties. These objectives are very close to the ones we want to achieve: modeling trust. This section aims at presenting how Verifpal is used in this work to model trust relationships. The first step consists in modeling the protocol by sending constants that correspond to the various messages sent by the different actors. Each constant sent by message will be "guarded" according to the trust that a certain actor has towards the other actor. For instance, if Alice sends something to her node, Alice trusts that node and the connection to it so the data will be guarded. This prevents any changes that a real attacker could not make to the data. When trust between two parties is not established, the attacker must be able to modify the data to be able to establish a counterexample if it exists. These trust relationships are variable depending on the point of view from which one is placed. From Alice's point of view, Bob's node is untrusted, so messages between Bob's node and the SoC should be unguarded. The properties to be checked can be represented using authentication queries. Indeed, if the attacker is able to modify unguarded messages without the protocol actors noticing it, then there is a vulnerability.

V. Trust Evaluation

The aim of this section is to describe the model and its verification based on the rules established in the previous sections. The modeling of three possible fixes, along with their drawbacks, is also presented.

A. State of The Art Protocol

The protocol proposed by Islam et al. [2] is presented in Figure 1. Listing 1 gives an excerpt of the Verifpal script that models this protocol. The upgrade request (lines 1–10) and data reception (lines 11–16) phases are shown. The first message transmission represents the request for SoC configuration modification and the adequate payment by Bob. To get this request written into the Blockchain, Bob uses his own node, which broadcasts it to all other nodes (including Alice's node). The actions performed by the Blockchain are represented at line 8 in the code by a hash operation on all the constants it has received, namely: the list of available configurations (previously initialized by Alice), and the state of the configuration requested by Bob. When the SoC makes a request to retrieve its configuration state (line 11), it is this hash that is provided to it, as well as the data it needs to check it. The check is performed line 15 with the `ASSERT` primitive, which compares the hash provided by Bob's node to the SoC's own hash reconstruction. If an attacker is able to fool the SoC, then this highlights a flaw in the protocol or in our model. Verifpal will verify this thanks to the authentication query. This query is used to ensure that an attacker is not able to modify the content of the message sent to the SoC. The execution of this Verifpal script gives the output shown in Figure 2. The execution of this Verifpal model gives a counterexample. It shows that an attacker is able to create an alternative version of the Blockchain content that will considered as valid by the SoC. By extrapolating, it is possible to conclude that a malicious entity could unlock any configuration without paying for it. Indeed, the data sent by Bob is not trustworthy from the SoC's point of view: the

```
Verifpal • Verification completed for 'tmp.vp' at 01:40:16 PM.
Verifpal • Summary of failed queries will follow.

  Result • authentication? Bob_node -> Soc: tamperproofdata — When:
           updaterequestandpayment → nil ← mutated by Attacker (ori
ginally updaterequestandpayment)
           unnamed_0 → ASSERT(HASH(updaterequestandpayment, ownersh
ipinfo_config_price), HASH(nil, ownershipinfo_config_price))?

           In another session:
           updaterequestandpayment → nil ← mutated by Attacker (ori
ginally updaterequestandpayment)
           tamperproofdata → HASH(nil, ownershipinfo_config_price)
← mutated by Attacker (originally HASH(updaterequestandpayment, owne
rshipinfo_config_price))
           unnamed_0 → ASSERT(HASH(nil, ownershipinfo_config_price)
, HASH(nil, ownershipinfo_config_price))?
           tamperproofdata (HASH(nil, ownershipinfo_config_price)),
 sent by Attacker and not by Bob_node, is successfully used in ASSER
T(HASH(nil, ownershipinfo_config_price), HASH(nil, ownershipinfo_con
fig_price))? within Soc's state.

Verifpal • Thank you for using Verifpal.
```

Figure 2. Trust issue counterexample proposed by Verifpal

SoC access to the Blockchain depends on Bob's node. The SoC cannot naively trust Bob's node to retrieve data from the Blockchain, because it is possible for that node to falsify the payment data for a new configuration.

B. Possible Fixes

We have seen that accessing Blockchain data from the SoC cannot be direct, and should not only depend on a subvertible node. In order to address this issue, this section presents possible fixes, all of which have their own limitations.

1) Trust The Node: The simplest fix is to have the SoC trust the data sent by Bob's node. In Verifpal, this means modifying line 12 of Listing 1 into `Bob_node -> SoC: [data], [updtRequestAndPay]`. The data is no longer vulnerable to attacks, and Verifpal can no longer find a counterexample. The problem with this solution comes from the consequences of establishing this trust relationship between Bob's node and the SoC. For Alice to trust Bob's node, she must be its owner, or at least control it. However, in this situation Bob may be harmed because Alice is able to refuse his configuration requests or even to scam him. We could also imagine that Alice and Bob agree to use a node from a trusted third party. But in this situation, the benefits of decentralization is lost.

2) Add a Contract-like Signature System: One way to allow the SoC to trust the data received from Bob's node is to have Alice countersign Bob's reconfiguration requests. Figure 3 presents the sequence diagram showing one way to achieve this. Listing 2 shows the Verifpal code that models the upgrade request (lines 4–20) and data reception phases (lines 21–28). With this protocol, when Bob makes a configuration request via the Blockchain, the payment made to Alice is blocked. Alice is notified and generates the certificate that unlocks the configuration. By sending this configuration via Blockchain the payment can be unlocked (the Blockchain is able to verify the certificate). Bob is able to retrieve the certificate and can provide it to the SoC. Finally, the SoC can verify the certificate and configure itself accordingly.

When verifying this model Verifpal does not find any

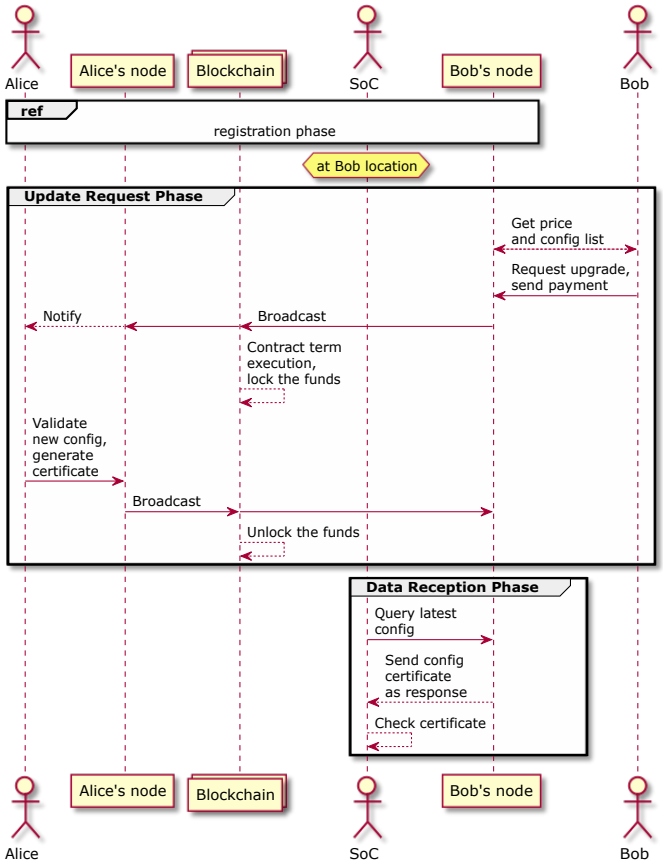

Figure 3. Sequence diagram of the signature based fix

possible attack. The limitations of this approach come from the fact that Alice must generate certificates. We therefore lose part of the automation allowed by smart contracts. This implies that Alice is able to refuse to validate configurations requested by Bob. Although Alice is not able to steal Bob (i.e., accepting payment without validating), she can prevent Bob from using his SoC in the way he wanted to.

3) Make the SoC Verify Blockchain Data: One way to allow the component to access configurations while not depending on Alice's endorsement is to make the SoC have direct access to the Blockchain as shown in Figure 4. This implies that the SoC is a node of the Blockchain in its own right. Technically this approach is hard to achieve. This is because it is necessary for the component to maintain and verify the Blockchain itself. However, this requires too much resources in terms of computing, memory and network [7] for the low-powered components that SoCs are. In the absence of the ability to act as a node, partial verification might be feasible. In a proof of work (PoW) [4] Blockchain like Bitcoin or formerly Ethereum, it should be possible to verify an affordable number of blocks. This number should be high enough that the cost of forging these blocks significantly exceeds the cost of purchasing the SoC configurations. Modeling data recovery via such an approach is presented in Listing 3. In this model, a single block is considered, as a simplification. The request

Listing 1. Excerpt code of the upgrade request and the data reception phase

```
// Upgrade request
Bob -> Bob_node: [updtRequestAndPay]

principal Bob_node[]
Bob_node -> Blockchain:[updtRequestAndPay]

principal Blockchain[
    data = HASH(updtRequestAndPay, chipInfoAndPrice)
]
Blockchain -> Bob_node: [data]
// Data reception
Bob_node -> SoC: data, updtRequestAndPay

principal SoC[
    _ = ASSERT(data,HASH(updtRequestAndPay, chipInfoAndPrice))?
]
queries[
    authentication? Bob_node -> SoC: data
]
```

Listing 2. Verifpal signature-based fix excerpt code.

```
principal Bob[
    knows private updtRequestAndPay
]
// Upgrade request
Bob -> Bob_node: [updtRequestAndPay], [pkAlice]

principal Bob_node[]
Bob_node -> Blockchain: [updtRequestAndPay], [pkAlice]
Blockchain -> Alice_node: [updtRequestAndPay]

principal Alice[
    generates certificate
    signed_certif = SIGN(skAlice, certificate)
]
Alice -> Blockchain: [signed_certif], [certificate]

principal Blockchain[
    _ = SIGNVERIF(pkAlice, certificate, signed_certif,)?
    data = HASH(signed_certif, chipInfoAndPrice)
]
// Data reception
Blockchain -> Bob_node: [data], [signed_certif], [certificate]
Bob_node -> SoC: data, signed_certif, certificate

principal SoC[
    _ = SIGNVERIF(pkAlice, certificate, signed_certif)?
    _ = ASSERT(data, HASH(signed_certif, chipInfoAndPrice))?
]
```

isn't present in the listing and lines 1–5 present the addition of a new block in front of the one that contains the updates. At data recovery time (lines 6–13), the upper block is guarded in order to represent the fact that it is not forgeable. The SoC verifies (lines 11–12), that the block that contains the update is not forged because it is protected by the new block that represents the most up-to-date state of the Blockchain. Verifpal does not find any counterexample with this model, which means that this protocol could be effective. The main drawback of this approach is that it can only work with proof of work Blockchains. However, this type of Blockchain presents some scalability issues [16], and has a very high energy consumption [17]. This is why Ethereum recently switched to proof-of-stake [18].

VI. Threats to Validity

There are several points in our approach that may be critical to the validity of our results. First, the used tool is not perfect. Indeed, Verifpal is still beta software, and the author does not guarantee that verification is exhaustive [6]. The models are nonetheless simple enough to have high confidence in the abilities of Verifpal to verify them. Second, the Verifpal models have been simplified to the extreme, and mainly express the trust relationships. This means that one could blame them for pointing out the obvious. In the case of the flaw demonstration in the studied protocol, this can indeed be the case, as the counterexample can be found without tools. However, the validity of the proposed fixes is, in this situation, more difficult and the use of a protocol verification tool is an improvement. Finally, it is not impossible that the flaw we have found is simple enough for the authors of the original protocol to consider it relatively easy to avoid.

VII. Conclusion and Perspectives

In this paper, we discussed a Blockchain-based protocol proposed in the literature [2] to enable SoC reconfiguration.

Listing 3. Verifpal Blockchain verification fix excerpt code

```
principal Blockchain[
    data = HASH(updtRequestAndPay, chipInfoAndPrice)
    generates newBlocktransaction
    newBlock = HASH(data,newBlocktransaction)
]
Blockchain -> Bob_node: [data], [newBlock], [newBlocktransaction]
Bob_node->SoC: data,updtRequestAndPay, [newBlock], [newBlocktransaction]

principal SoC[
    // Upper block verification
    _ = ASSERT(newBlock, HASH(data,newBlocktransaction))?
    _ = ASSERT(data,HASH(updtRequestAndPay, chipInfoAndPrice))?
]
```

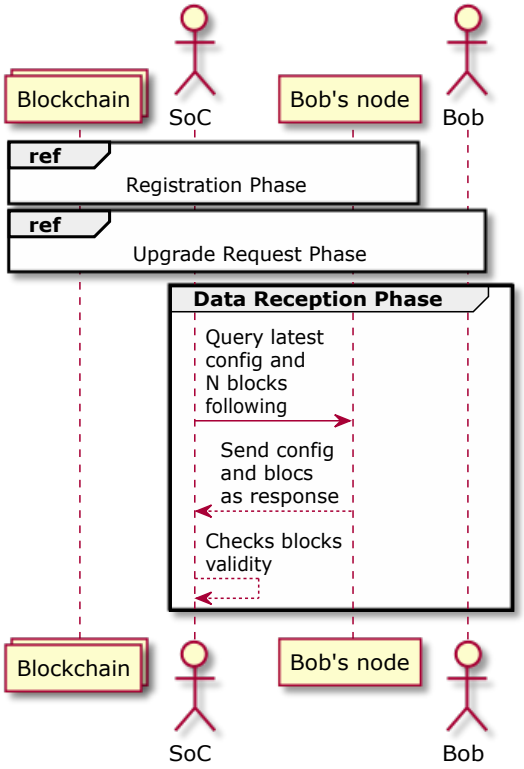

Figure 4. Sequence diagram of the block verification based fix

We showed, by modeling the protocol with Verifpal [6], that its current design has a flaw on the SoC data access. We proposed several possible approaches to fix this flaw, at the cost of losing some Blockchain properties. Among these solutions, two stand out. A first one that solves the problem through a decentralized certificate exchange system. The main problem is that it requires the approval of both parties regularly, which implies potential censorship by one of the actors. A possible solution could be a system of sanctions and/or compensations to push the actors not to behave in this way. A second solution proposes to allow the SoC to partially verify the Blockchain. In this situation, the security guarantees on the validity are weaker than with direct access to the Blockchain, but if the forging cost is more important than the component value, it can be considered as equivalent. The main problem with this solution is the need to use a proof of work Blockchain, which is energetically expensive. Our approach has several limitations. Our models are relatively simple and could be extended to more accurately model the Blockchain. Moreover, Verifpal is not directly designed for this kind of verification, but it is relatively easy to learn and use. Possible extensions of this work include: the use of a more appropriate modeling language to represent trust. A more precise version of the protocol model could also be considered. One of the possible future works would be the implementation of one of the proposed fix, for example: the certificate exchange system with a proof of concept in solidity, the Ethereum dedicated smarts contracts language.

Acknowledgements

The authors would like to thank Nicolas Pouillard for the invaluable discussions and advice on topics related to this work.

References

[1] M. Mere, F. Jouault, L. Pallardy, and R. Perdriau, "Trustworthy SoC Reconfiguration Aimed at Product-Service Systems: a Literature Review," in *2022 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*. Barcelona, Spain: IEEE Computer Society, Aug. 2022, pp. 1–6. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/COINS54846.2022.9854965

[2] M. N. Islam and S. Kundu, "Remote Device Management via Smart Contracts," *IEEE Transactions on Consumer Electronics*, vol. 68, pp. 38 – 46, 2021.

[3] J. Arcenegui, R. Arjona, R. Román, and I. Baturone, "Secure Combination of IoT and Blockchain by Physically Binding IoT Devices to Smart Non-Fungible Tokens Using PUFs," *Sensors*, vol. 21, no. 9, pp. 1 – 23, 2021.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008, 2020-07-22. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[5] B. K. Mohanta, S. S. Panda, and D. Jena, "An Overview of Smart Contract and Use Cases in Blockchain Technology," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. IEEE, Jul. 2018, pp. 1–4.

[6] N. Kobeissi, G. Nicolas, and M. Tiwari, "Verifpal: Cryptographic Protocol Analysis for the Real World," 2019, published: Cryptology ePrint Archive, Report 2019/971. [Online]. Available: https://ia.cr/2019/971

[7] ethereum.org, "Nodes and clients," Jun. 2022. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/#hardware

[8] Y. Ruan and A. Durresi, "A survey of trust management systems for online social communities – Trust modeling, trust inference and attacks," *Knowledge-Based Systems*, vol. 106, pp. 150–163, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705116301393

[9] M. H. Miraz, "Blockchain of Things (BCoT): The Fusion of Blockchain and IoT Technologies," in *Advanced Applications of Blockchain Technology*, S. Kim and G. C. Deka, Eds. Singapore: Springer Singapore, 2020, pp. 141–159. [Online]. Available: https://doi.org/10.1007/978-981-13-8775-3_7

[10] G. Caldarelli, "Overview of Blockchain Oracle Research," *Future Internet*, vol. 14, no. 6, 2022. [Online]. Available: https://www.mdpi.com/1999-5903/14/6/175

[11] H. Sukhwani, N. Wang, K. S. Trivedi, and A. Rindos, "Performance Modeling of Hyperledger Fabric (Permissioned Blockchain Network)," in *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, Nov. 2018, pp. 1–8.

[12] H. Rocha and S. Ducasse, "Preliminary Steps Towards Modeling Blockchain Oriented Software," in *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, May 2018, pp. 52–57.

[13] B. Blanchet, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016. [Online]. Available: http://dx.doi.org/10.1561/3300000004

[14] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN Prover for the Symbolic Analysis of Security Protocols," in *Computer Aided Verification*, N. Sharygina and H. Veith, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 696–701.

[15] I. Cervesato, "The Dolev-Yao intruder is the most powerful attacker," in *16th Annual Symposium on Logic in Computer Science—LICS*, vol. 1. Citeseer, 2001, pp. 1–2.

[16] D. Khan, L. T. Jung, and M. A. Hashmani, "Systematic Literature Review of Challenges in Blockchain Scalability," *Applied Sciences*, vol. 11, no. 20, pp. 1 – 27, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/20/9372

[17] J. Sedlmeir, H. U. Buhl, G. Fridgen, and R. Keller, "The Energy Consumption of Blockchain Technology: Beyond Myth," *Business & Information Systems Engineering*, vol. 62, no. 6, pp. 599–608, Dec. 2020. [Online]. Available: https://doi.org/10.1007/s12599-020-00656-x

[18] F. Saleh, "Blockchain without Waste: Proof-of-Stake," *The Review of Financial Studies*, vol. 34, no. 3, pp. 1156–1190, Jul. 2020, _eprint: https://academic.oup.com/rfs/article-pdf/34/3/1156/36264598/hhaa075.pdf. [Online]. Available: https://doi.org/10.1093/rfs/hhaa075