Managing Identities Using Blockchains and CoSi

Eleftherios Kokoris-Kogias, Linus Gasser, Ismail Khoffi,
Philipp Jovanovic, Nicolas Gailly, Bryan Ford
EPFL

1. INTRODUCTION

Nowadays we have to identify ourselves to different services and devices, either by a login/password or in the case of SSH, with a public key. Often the same public-key is registered with different services and thus it is easy to track the user across different platforms. To enhance privacy, the user should use a separate SSH-keypair for each service, but this is difficult to manage. The goal of our system is to enable the user to use multiple accounts across his devices and to relieve some of these difficulties. This is applicable in multiple systems like PGP [2], or SSH [9], but we focus for the moment on SSH because it is widely-used both for connecting a user's different machines (via remote login and scp) and for identifying the user (e.g., github). Specifically, we address the cases where the user (a) rotates keys for security purposes, (b) introduces new key-pairs in her online-presence, and (c) revokes key-pairs that she can no longer access.

The most straightforward way to use SSH is to create one SSH-key and copy it to every other device. However, controlling this one key enables an attacker to compromise all the user's devices. On the contrary, the recommended practice [6] is to create one key per host and to limit the damage an attacker can do when compromising one of the user's devices, by configuring the key's privileges to be a subset of the root privileges. A drawback is that this practice requires a user to update the access lists of numerous servers, or to delegate this to centrally managed services [1].

We combine collective signing [8] and blockchains [5, 3] to create a secure and easy-to-use, decentralized SSH-key management system. The SSH management is done by a cothority [8] that maintains a list of authoritative keys of the user. Whenever a change is needed, due to key rotation or lost/new devices, the user initiates an update where the cothority contacts the devices that control the keys and gets a signed approval of the new block. If a threshold (typically three or four) of devices agree on the change then the cothority transmits the individual signatures and the block which is then collectively signed by the cothority. This signature signifies the agreement of the cothority on witnessing enough proof that the keys should change and models a *forward link*, making the blockchain doubly-linked.

2. SYSTEM ARCHITECTURE

Figure 1 illustrates the architecture of the system. The *Manager-Devices* are devices that the user can physically access, like laptops, and phones. They are authoritative for managing the user's identity by proposing new data to be included in the blocks. In our use-case, this data consists of public SSH-keys. However, the managers are not always available and/or accessible, thus they cannot be used to serve new blocks to the clients, or manage the distributed coordination. To mitigate this problem we assume the existence of publicly reachable servers forming a cothority [8], that provides blockchain management as a service. Whenever the user wants to add, remove or rotate some keys she creates a new update-request that is sent to the *Cothority-Servers* and gets signed by a threshold of managers (creating a forward link). Finally, *Clients* are remote servers, IoT devices, or services (e.g. github) that the user wants to access. They download and verify blocks from the cothority in order to automatically update their access lists. If they are online they are notified at the moment of creation, otherwise they randomly poll a cothority server when they get online.

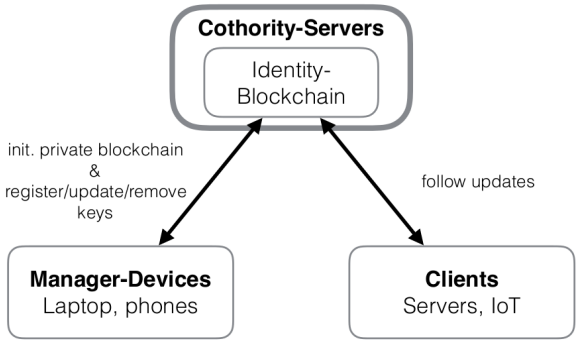

Figure 1: System architecture

The basic data-structure of the system is the *Identity-Blockchain*, which is made of blocks that store the SSH-keys of the user's devices. The blockchain is doubly-linked, its backward links are used for ordering between the blocks and are cryptographic hashes of the previous block. Future blocks do not exist at the time of block creation, thus forward links cannot be represented as cryptographic hashes incorporated into the block's hash (like backward links). They are retroactively added inside the block. In order to enable the secure creation of forward links we employ digital signatures. Whenever a new block is requested the cothority creates a new block containing the new data and a backward link to the current head. Afterwards, a threshold of managers sign the new block with their current key thereby creating a forward link from the current to the new block, which makes the latter also the new head of the chain. That way, they delegate the trust from their old keys to the new ones and a client following the chain can easily verify each step it is taking, without depending on an intermediary for the validity of the retrieved information.

3. SECURITY CONSIDERATIONS

Our system enables the user to follow best practices [6] with her SSH-keys without the need to manually manage them or completely outsource their managment. Existing, centralized solutions create targets for attackers as they allow them to directly inject new public SSH-keys. Our solution has no central authority that can be abused, and even if the cothority is compromised they can only DoS the system, because they are unable to produce forged blocks without holding a threshold of the managers' private keys. Further-

more, an attacker holding only one private SSH-key is not able to modify the access list of the clients, since he needs a threshold of these keys to propose a valid new block. We can further enhance the security of the system by introducing a two-factor login, especially for keys with increased rights, so that even in the case of a compromised manager device the attacker would not be able to log in to any of the clients.

One attack the system is vulnerable to, is a freeze-attack on behalf of the cothority, but this is mitigated by the fact that the servers holding the blockchain are randomly chosen. We also define a maximum epoch of a block, so that if a client sees that there is no new block, he can assume a freeze-attack is happening and take necessary actions.

Finally, we expect most of the manager-devices to not be accessible via SSH (which is the case for phones and tablets), but if a device is accessible via SSH, the private keys of the device should be password protected, so that the attacker cannot remotely open them and control the corresponding private key of this device too.

4. IMPLEMENTATION & EVALUATION

Our implementation of the cothority[1] supports all operations described here. It has been tested to run with up to 32'000 instances and can sign new blocks in a matter of seconds. In practice different subsets of the cothority will be chosen to handle different identity-blockchains. The manager-devices are handled by a program that implements all functionalities needed to communicate with the identity-blockchain service on the cothorities. A manager-device proposes a change to the blockchain which is transmitted by the cothority to the other manager-devices and has to be confirmed by a threshold of them. The clients have a list of identity-blockchains that they follow. In the case where a client missed several updates, he will receive all blocks necessary to prove that the update is trustworthy.

Every manager-device that wants to confirm a change needs to download the list of public keys, which is up to 1kB. The confirmation-signature on this list is about 64 bytes and has to be sent to the cothority. Once the cothority has signed a new list and appended it to the blockchain, there is an overhead of 256 bytes for the hash, the signature, and other configuration-fields for the blockchain. When a client downloads the new list it also has to download that overhead and then needs to verify that the signature is correct. This verification uses a Schnorr-signature with the aggregate public key of the cothority and as such is lightweight, even in the case of a large number of cothority-servers. The overall overhead is minimal and the system can be used by resource constrained devices (e.g., in IoT), too.

5. FROM SSH TO IDENTITY MANAGEMENT

We implemented the SSH-Key Management as an illustrative example, but we envision a larger Identity Management system, where a user can manage not only his SSH-keys, but also PGP keys or keys that can be used to contact him in an end-to-end encryption messaging system, without delegating the management of them to the messaging platform's operator.

We are currently implementing such an identity management system based on CONIKS [4], but changing it from a name-to-key resolver to a name-to-distributed-identity resolver. A user can provide the ID of his blockchain and a way to retrieve it (e.g., the IP address of the cothority). Anyone wanting to contact the user can then retrieve the block and verify its validity (like the clients of Fig. 1 do). Furthermore, he can request and verify all the intermediate blocks, between the last known identity of the user and the one retrieved, to verify consistency. As far as privacy is concerned, we use all the privacy features of CONIKS (the keys are hashed so a client can retrieve a block only through knowledge of the keys, the ID can be encrypted by a symmetric key if the user does not want to be publicly reachable, etc.). Additionally, we add the option that parts of the block can be encrypted with another symmetric key so that someone could communicate with the user leveraging the publicly available key but would need to request the symmetric key to get more sensitive information out. However, once he has this key, any updates to these parts are easy to follow; there is also a next-key field so that the user can rotate symmetric keys efficiently. Finally, we have already added Collective Signing [8] on the STRs of the Identity Providers, so that hashchain forks are pro-actively eliminated and the user is fully secured without contacting multiple auditors (which can be hard in censorship-prone locations).

With the above system we are planning to build, easy-to-use PGP messaging. We decouple the blockchain's uses to *internal*, for updating access lists, and *external* for following the identities. The cothority helps the user manage per-device SSH/PGP keys internally, and collectively generates the public-facing PGP-key (which is used to contact the user), using threshold signatures. This key is created via secret sharing [7], and a threshold of servers are needed to reconstruct it. Now each device not only has one SSH-key, but also one PGP key, while the user publicly announces the PGP key of the cothority. When a PGP encyrpted message arrives, the user authorizes its decryption by signing approval with an internally used key. However, if a manager-device is compromised, the attacker can temporarily use that to authorize the signing or decryption of messages until the attack is detected. Once detected the attacker loses access, and the composite public-facing PGP key remains uncompromised. This increases both security of private communications, since the PGP key is not copied to every device, and usability, since a lost device does not lead to the key's rotation.

For privacy-conscious users the same method can be used for SSH-keys, as well. They can have their actual SSH-keys only for internal use and create multiple external blockchains, that contain cothority-produced keys. This way they will be able to use the cothorities as a proxy to connect to different services pseudonymously and make it harder for colluding services to track their online actions.

[1] https://www.github.com/dedis/cothority

6. REFERENCES

[1] Authentication System privacyIDEA, May 2016.

[2] M. Elkins. Mime security with pretty good privacy. 1996.

[3] E. Kokoris-Kogias et al. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *25th USENIX Conference on Security Symposium*, 2016.

[4] Melara et al. CONIKS: Bringing key transparency to end users. In *24th USENIX Conference on Security Symposium*, 2015.

[5] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.

[6] NIST. *Security of Interactive and automated Access Management Using Secure Shell (SSH)*, chapter 5.1.2. 2015.

[7] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[8] E. Syta et al. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *37th IEEE Symposium on Security and Privacy*, 2016.

[9] T. Ylonen and C. Lonvick. The secure shell (ssh) protocol architecture. 2006.