Poster: Decentralizing PKI with SDSI Keychains

Harry Halpin
Inria
Paris, France
harry.halpin@inria.fr

***Abstract*—One of the key problems facing security has been a coherent way to refer to and access key material in a decentralized fashion, given the many usability failures of traditional PKI schemes based on X.509. In this concept note, we propose re-purposing the access control scheme SDSI (Simple Distributed Security Infrastructure) for use with blockchains (i.e. append-only logs of key material). SDSI is a long-standing and well-studied alternative to the hierarchical X.509 infrastructure, but failed to be adopted due to problems with key discovery and revocation, both of which can be solved using blockchain technology.**

I. The Public-Key Infrastructure Problem

The problem of Public Key Infrastructure (PKI) is how to bind an identity of a principal (user, organization, etc.) to a key. This "PKI problem" has been one of the most long-standing and hardest problems facing the adoption of public key cryptography. The X.509 certificate infrastructure operates in theory as a centralized key directory, and in practice the X.509 standards were added post-hoc to the centralized domain name system (DNS) and TLS via Certificate Authorities. The X.509 PKI system has been thought of as a failure, as compromises in the certificate authority system have led to fake certificates being issued without being detected. For emerging technologies based on public cryptography, including blockchain technologies and secure messaging, the problem of binding a key to a user has been solved in an ad-hoc manner.

It may very well be time for new technologies to revisit the road not taken by the Web: the Simple Distributed Security Infrastructure (SDSI) of Rivest and Lampson. Similar to Bitcoin, principals are identified with keys, in particular with digital signature keys. Also similar to blockchain technologies but in contrast to X.509, each principal is a "certification authority" where "certificates can be created and signed by anyone"[4]. Unlike X.509 PKI, local name-spaces can be created where a principal may associate any arbitrary naming convention with a key. These local names can be explicitly linked across namespaces without centralized permissions. Although the failure of SDSI has been attributed simply to various accidental factors (i.e. it being produced by academics after X.509), there is a outstanding problem with SDSI: Due to its decentralized nature, it was impossible to tell if a key was the latest key, or if a key had been revoked. Key revocation and rotation were considered to be dealt with by "self destruct" or "key update" messages that invalidate keys for a given identifier or bind them to new identifiers, but this work was left undefined [1].

II. SDSI 1.0

Traditionally, SDSI can be defined as the following: Given K as the set of public-private key-pairs ($K_i = (\mathsf{pk}_i, \mathsf{sk}_i)$) and a set A of identifiers (usually strings), in SDSI a **local name** is a map between an key and one or more identifiers ($K \Rightarrow A$). A certificate is a binding between values and a name, is a signed tuple (K, A, M) where M provides any additional metadata, including but not limited to the validity period of the key. K is the keypair used in signing the certificate. A valuation function for a given identifier A_j is $V(A_j) = K$, a (possibly empty) set of public keys for a given term. Valuation can be defined and implemented as a set of rewrite rules [1]. These rewrite rules allow for a given key the discovery of all associated identifiers for a given key or for a given key to find all associated identifiers and certificates, but these rules simply find all keys. As the set of keys found by $V(A_j)$ could be potentially very large and there is no necessary connection between keys other than possible metadata in M, in practice SDSI did not deal with key handling, discovery, revocation, expiration, and rotation.

SDSI allows the export and linking of local names. For example, (K_1 `Nakamoto`) binds public key K_1 to the identifier `Nakamoto` in a name space (such as Alice's namespace), perhaps including additional information in a certificate such as (K_1, `Nakamoto, creator of Bitcoin, 31-8-2008`). Then Bob can make statements about who Alice thinks Nakamoto is such as (`Alice's Nakamoto Szabo`). SDSI can export (`Bob's Nakamoto`) to refer to an entirely different key (`Bob's Nakamoto Adam`) where (K_2 `Adam`). The identifier `Nakamoto` can be used by any other namespace, such as Eve, to identify another key (K_3 `Nakamoto`) or to associate more identifies with the identifier .

Access control is the motivating use-case of SDSI. Traditionally, SDSI defines groups as a set of principals. A group as such does not have its own key, and each member of the group may offer their key as a proof of their membership in the group. Classically groups can be defined via use of the reserved term *Group* and logical AND, OR, NOT as well as ALL, $MINUS$

(for exclusion of principals) and ANY. For example, `(DAO's decision-makers ( Group: ( OR: eth-core-dev ( AND: investor boardmember )))` defines a group for the DAO's decision-makers where one must be either an Ethereum core developer or an investor and board member to make a decision on a smart contract.

III. Using Keychains to Support SDSI

SDSI keychains is the use of SDSI where a simplified blockchain of keys rather than keys are the principal. In terms of blockchain technologies, only a simple authenticated append-only list of keys that is authenticated via hash pointers is needed, with the key of each block signing the previous block. This design has been called *claimchains* in a general framework [2]. This use of a claimchain for a *keychain* is similar to the more complex work put forward in CONIKS [3], where each keychain incorporates a Merkle Tree so that verifying the presence of a key in a keychain can be done efficiently, and the *head imprint* (i.e. the hash of the latest key) can witness the state of an entire keychain. These head imprints may also be stored inside other keychains so that statements may be made about these heads in order to make statements about another keychain at a given moment in time, allowing the *linked local namespaces* of SDSI.

Given standard cryptographic definitions, a SDSI tuple $S_i = (K, A, M)$ and its hash $H(S_i) = H(K \parallel A \parallel M)$ and M must include t, time of tuple creation. A **keychain** is an ordered sequence of *keyblocks* $B = \{B_0, B_1, ..., B_n\}$. Each keyblock $B_i = (S_i, P_i, \sigma_i)$ in our chain comprises a SDSI tuple S_i, and a set of hashes of previous blocks P_i and a signature $\sigma_i = \mathsf{SIG}_{\mathsf{sk}_i}(H(H(S_i) \parallel P_i))$ with sk_i of K_i. Keyblocks so have a global strict ordering as defined by the index i, with the latest keyblock $i = max$ so B_{max} has a unique key K_{max}. All statements A are assumed to apply to the principal defined by B unless the statement is explicitly revoked.

Keys can be revoked by including a revocation statement as part of $M = (K_{new}, R, t)$ where R is revocation made at time t with a new signing key for subsequent keyblocks given by K_{new}. Statements can be revoked by signing new statements later in the blockchain using an explicit revocation statement over the previous statement. The latest key can always be found at the head keyblock as each block has a single signing key, as well as proof of any key rotations and revocations via the aforementioned efficient search for older keys [1]. Statements range over entire histories of keys, rather than just keys, and blocks of statements (such as revocation statements or the addition of new keys) can always be authenticated. We still allow, like the original SDSI, $V(A_j)$ to result in multiple keychains, but they can be compared and a latest key always found by looking at the creation time t of the blocks in case multiple keychains with differing values of B_{max} are found. From this simple mechanism, the entire group-based access control design of SDSI using straightforward logical operators over key material can be rebuilt using keychains rather than keys. Unlike the original SDSI, given a single key, we can find the latest key and any new identifiers given in the authenticated history of this key.

IV. Next Steps

We've outlined the usage of SDSI over blockchains, where rather than keys being principals as in normal usage of SDSI, blockchains consisting of an append-only log of keys are given as principals. This gives a flexible way for metadata to be associated with the state of a given blockchain at a given time, and for blockchains to refer to each other. Using the well-explored space of SDSI name resolution, local names can then be resolved to the state of SDSI-enabled keychains, and the latest key in a given keychain can be used for cryptographic operations, with key revocation being recorded by the blockchain for a given key. We have only sketched the design, and full definitions with a formal semantics for access control is needed in future work. It is widely-accepted that the adoption of hierarchical public key infrastructure for the Internet was a mistake, and there's no reason for blockchain researchers have to repeat the mistakes of the past. With SDSI for blockchains, we can rely on a well-founded cryptographic framework for decentralized identity, not having to repeat the Web's mistakes.

Acknowledgment

This work was funded by European Unions Horizon 2020 Framework Programme for Research and Innovation (H2020-ICT-2015, ICT-10-2015) under grant agreement number 688722. This use of SDSI for access control derived from work-in-progress on "Claimchain" done by partners George Danezis, Bogdan Kulynych, Marios Isaakidis, and Carmela Troncoso for D2.2 of NEXTLEAP.

References

[1] Dwaine Clarke, Jean-Emile Elien, Carl Ellison, Matt Fredette, Alexander Morcos, and Ronald L Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[2] George Danezis, Bogdan Kulynych, Carmela Troncoso, Marios Isaakides, and Harry Halpin. Claimchains: A decentralized identity system based on hash chains, 2016.

[3] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. CONIKS: bringing key transparency to end users. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15*, pages 383–398. USENIX Association, 2015.

[4] Ronald L Rivest and Butler Lampson. SDSI-A Simple Distributed Security Infrastructure. CRYPTO, 1996. http://people.csail.mit.edu/rivest/sdsi10.html.