DID Recovery via a TimeStamping Service

Omer Shlomovits[*] Hicham Batou[†] Jakub Jastrzębski[‡]

Simple DID Model and Problem Statement: Our work is focused on a simplified model for decentralized ID (DID) system. Within our model we assume two types of actors:

1. *Clients*: physical owners of Identities.
2. *Nodes*: will form the DID network: responsible for message routing

The system security will be defined using a single signature key. This key, generated by the client in a unique and secret way, is equivalent to the client identity: The only way to make sure a statement is attached to an identity is by providing a signature on the statement that is verified using the signing public key. Our model supports only messages of type "statement" in which some identity is publishing a signed message to the network. While a real world DID system is much more complex, potentially including more types of actors and keys, our simple model is enough for a full exposition of our problem [1].

Our work is focused on the problem of signature key *recovery*. There are two threats here:

- Signature key is lost $\rightarrow$ the identity can no longer make new statements
- Signature key is stolen $\rightarrow$ The attacker now controls the identity

With an emphasis on simplicity we need to find a recovery protocol that adheres to the following requirements:

1. Create a new signing key
2. Revoke the old signing key
3. Continue to represent the same identity

In the rest of this abstract we will provide the high level details of such a protocol, based on timestamping service by the nodes. In addition we discuss concurrent practical solutions which we claim do not cover the full set of requirements.

Foundations of DID key recovery: Out of the requirements stated above, the first one is rather easy: The same mechanism used to to generate the main signature key will be used to generate a new backup key. The second and third requirements however are less trivial. We first present two observations that apply for the case of DID and will become crucial for our protocol.
Observation 1: Signing key is random while Identities are unique. In theory, we can generate an infinite number of signing keys that will represent the same identity.
Observation 2: An identity can use a signing key to make a statement on the identity itself.
Combining these two observations leads us to the optimistic conclusion that instead of recovering a key, we only need to recover a signed statement. As a first stab at a solution we suggest the following: What if instead of generating one signing key, we initially generate two signing keys. One will be the

[*]ZenGo-X
[†]Planetworks
[‡]Planetworks
[1]this may hint on the generality of our solution

main key and one will be the secondary or backup key. Immediately after key generation we sign with the main key on an EQUAL-ID statement as follows: *main public key: XXX and backup public key: YYY represents the same identity.* The backup key can now be kept at some offline secure storage together with the signed EQUAL-ID statement.

Handling malicious EQUAL-ID statements: A simple attack on the described solution is for an attacker to create a fake EQUAL-ID statement, and sign it with the stolen main signing key. How can the network tell who is the true "owner" of the recovered identity? Our solution therefore requires a third party timestamping service: Each statement issued using a signing key must be time-stamped in a way that other nodes/participants in the network can verify and ensure the chronological order of all statements. Under our model, the most natural approach for providing a timestamping service is by a committee of the nodes reaching consensus: This way we get robustness via fault tolerance and easy accessibility. Practically, since timestamping using nodes should be verifiable - the nodes must attach a digital signature to each timestamp. The signature can be a threshold signature to increase security and reduce space.

Timestamping create a natural order. Therefore the EQUAL-ID that the network will consider valid in case of two EQUAL-IDs statements for different backup keys (one honest, one malicious) is the EQUAL-ID with the earlier timestamp signed by the committee.

Timestamping EQUAL-ID : What we have so far is the EQUAL-ID statement signed by the main key and timestamped by the nodes at the time of the keys creation. There is still one issue left: We want to keep the signed EQUAL-ID statement a secret until we need to use the backup. This is because, once published, the EQUAL-ID will signal all nodes in the network to mark as *invalid* any statement signed by the main key that was issued in time after the EQUAL-ID timestamp. The problem: On the one hand we need to publish EQUAL-ID in order to timestamp it at the moment of the key's creation. On the other hand, we want to publish EQUAL-ID only once the main key is compromised. Here is how we solve this tension: At the moment of creating the keys, EQUAL-ID is generated and signed using the main key. However, instead of publishing it and asking the nodes to timestamp-sign it, the owner will publish a perfectly-binding cryptographic commitment to the signed EQUAL-ID statement. The commitment will be timestamped instead of the plaintext. A backup key is activated by publishing the decommitment together with the relevant messages and signatures. The decommitment, assume checked and verified, will be timestamped and will mark the point in time from which all statements coming from the main key are not to be trusted anymore.

Putting everything together: We now sketch the full protocol, show that it achieves the full set of requirements we defined and discuss some extensions. For our construction we assume the nodes are using threshold signing with key k_n

Backup

1. Client generates two signing key pairs: main k_m and backup k_b
2. Client signs using k_m on statement EQUAL-ID: k_m *and* k_b *represents the same identity* and outputs signature σ
3. Client generates a hash based commitment COM with randomness r and message $m = \texttt{EQUAL-ID}||\sigma$
4. Client publishes COM to the network
5. Nodes reach consensus about the timestamp ts to add to COM. The nodes publish a signed timestamped commitment: $\sigma_n = \texttt{SIGN}_{k_n}\{\texttt{ts}||\texttt{COM}\}$

Recover

1. To activate a backup key, the client publishes $\{\texttt{DECOM} = \{r, m\}, \texttt{ts}, \texttt{COM}, \sigma_n\}$
2. Nodes on the network check `DECOM` correctness with respect to `COM` and verify σ_n
3. If all checks pass: the nodes will now ignore all new statements that are signed by main key.

Full protocol (sketch)

Revisiting our requirements: A new signing key was generated, simultaneously with the main key, thus fulfilling our first requirement. The decommitment to the `EQUAL-ID` statement signals the network that a backup key is now the main key. Any statement signed with the main key is not valid anymore, thus fulfilling the second requirement. Finally, The backup key represents the same identity facing the network.

Practical considerations: When a backup key becomes the main key, a new backup key must be generated. This new key will take on the role of a new backup key. In general, many backups keys can be generated for the same key and the rule is that "old" backup key will win over "new". The backup string (Recover protocol line 1) should be kept safe, separated from the main active key. Offline device should suffice but a secret sharing scheme can be used to provide some robustness. Nodes run distributed key generation and can refresh their secret shares of k_n to get proactive security.

Related work: We focus on three alternative designs: Tor.us[2], Signal[3] and CanDID[4] Tor.us is a secret-sharing based key management system, Signal is a messaging platform with end to end privacy and CanDID is a DID system based on a committee running multiparty computation. We argue that none of these systems handles recovery from identity theft in a satisfactory manner: Tor.us does not offer any protection when a key is in use - an attacker can extract the ID owner key and nothing prevents it from using it. Signal recovery depends on a short PIN number, while they make it hard for an outside attacker to make a lot of guesses about the PIN, the case of an attacker who gets hold of the PIN is not covered and it is unclear how the true ID owner can do revocation. CanDID handles key theft. Informal argument suggests that the true owner can provide a new ID with more attributes than it had before which will convince the committee to revoke the attacker controlling the old master credentials. However, this creates a tradeoff - a "strong" identity will want to have as many attributes as possible, however, in that case, recovery from theft becomes harder.

Talk structure: The context of this work is of a larger, more involved, new system for DID. The Planet system aims for simplicity which is what motivated us to explore the concept of recovery for secure IdM. We will start our talk by providing said context and motivation which will serve as proper background to DID systems. To complete the prerequisites We will explore the known and related methods for key management and key recovery. We will then present our protocol, in a way that follows the presentation in this abstract: requirements, fundamental new assumptions, first attempt at a solution, refining the solution, optimizing, practical considerations. By the time of the workshop we plan on having a full blown system for demonstration.

[2]https://docs.tor.us/how-torus-works/overview
[3]https://signal.org/blog/secure-value-recovery/
[4]https://eprint.iacr.org/2020/934.pdf