Report on interoperability between the Ethereum and Secure Scuttlebutt networks

This document follows the groundwork from the ethresearch article'Implementing secp256k1 on Secure Scuttlebutt to create cross-platform Ethereum-Scuttlebutt applications'.

Note:

Links had to be removed from this post, so it is missing some references.

Abstract

The Secure Scuttlebutt protocol was initially developed for a peer-to-peer social network. While this is still it's best known and most popular use-case, it is being adopted for a growing number of other applications, and it's relationship-centred nature gives it some interesting properties which make it distinct from distributed-hash-table based peer-to-peer protocols.

We discuss SSB's application as an off-chain transport and storage solution for Ethereum Dapp developers, and present a proof-of-concept for introducing secp256k1 signing to SSB. SSB is suited to applications which center around relationships, such as state channels, or sending partially signed transactions between co-signers, but less appropriate for applications which center around large scale content distribution.

Comparison of SSB with other off-chain protocols

Blockchains are suited to information where widespread consensus is critical, but due to the associated costs and limitations, it makes sense to use them as little as possible and have well-integrated off-chain systems. Ethereum is not simply a blockchain protocol but a stack of protocols including some which are designed specifically for off-chain transport and storage. So it makes sense to look at these protocols in order to establish whether SSB has something to offer which is not already there.

Whisper

Whisper is a communication protocol with a focus on ephemeral messaging. SSB, however, is designed to be a permanent, immutable log, although ephemeral messages can be sent over SSB using the module 'SSB-ephemeral-keys'. For this reason, Whisper is better suited to 'off-the-record' messaging, where data persistence is not important.

Whisper messages include the recipient as metadata, meaning an external observer can see who is talking to who. SSB-private obfuscates recipients, meaning an observer can see who is publishing encrypted messages, but not who they are being sent to. This is only practical to do because of SSB's gossip protocol. There is no ubiquitous view of the network, rather each peer can only access messages from a specified number of 'hops' away from their own node on the social graph. This means each peer has a small enough sub-set of the network that attempting to decrypt all messages because practical. This might seem like a lot of effort to go to just to obfuscate some metadata, but metadata leaking has been one of the biggest criticisms of traditional encrypted messaging approaches like PGP over email.

Whisper uses a DHT which may make it vulnerable to eclipse attacks, whereby a specific node is targeted to make part of the network inaccessible. SSB's gossip protocol makes these kind of attacks more difficult.

Swarm

Swarm is a distributed storage platform and content distribution service. SSB also offerers distributed data persistance, but it is designed for messages rather than large files. While SSB does have a system for requesting and distributing larger binary 'blobs', the protocol is not specifically designed for large files or data sets.

Proposed approach

Lets consider two cases where interoperability between SSB and Ethereum would be useful:

1. You want to interact with some entity with a known Ethereum address using the Scuttlebutt protocol. Eg: by referencing in a Scuttlebutt message
2. You want to interact with some entity with a known Scuttlebutt feed id using the Ethereum protocol. Eg: by sending them funds or including them in a smart contract.

The problem is that the two systems use a different elliptic curve, so currently there can be no direct cryptographic relationship between an Ethereum address and an SSB feed id. SSB was designed to be cryptographic-primitive agnostic, so all feed ids and signatures have a suffix which describes them, typically '.ed25519', and similarly references to messages and blobs have a suffix describing their hash function. So we propose to introduce feed ids and signatures on the

secp256k1 curve.

There is of course the difficulty of Diffie-Hellman encryption between keypairs on different curves. For this reason, we propose to only use secp256k1 keys for signing and have an associated curve25519 keypair for encryption.

However, it is worth noting that there could be a further use case for cross platform inter-operability where we didn't have this problem. Suppose you want to use the Scuttlebutt protocol for off-chain communication using Ethereum accounts as identities, but were not interested in existing Scuttlebutt users, messages or infrastructure such as pubs. Establishing a new SSB network where another curve is the norm would be not so difficult. But for now, we assume that we need to design a system that allows for existing users to interact with users of the new cross-platform system.

EIP 712 - Ethereum typed structured data hashing and signing

We propose to conform to EIP712, a specification of typed, structured data which means that the same structured data reliably gives to same hash or signature. This has implications for interoperability because it mean that signed information replicated over SSB could be directly included in a contract at a later time. We propose to use Metamask's node module for signing and verifying with EIP712, 'eth-sig-util'.

Current state of this project

- Key generation, seeded key generation, signing and verification are implemented and passing tests on ourfork of ssb-keys.
- We have used 'secp256k1-node', and the 'cryptocoinjs/keccak' module, the same one as ethereumJS, for hashing messages before signing.
- The .secp256k1

suffix for feedIds is allowed in our fork of ssb-ref

- Some other experiments can be found in our repo 'blockades/secp_experiments'

Using these forked modules, we were able to publish messages like the one below to a test network.

Example message published with a secp256k1 feedId and signature:

{ key: '%vtCIO56flJjvnokHmM0Fl2WXcITNaLV9RnQzTl7edyM=.sha256', value: { previous: '%YFhNH3h97zkS/LTzo/EImAP9Nyl3KNBVxSI5HE77TBU=.sha256', sequence: 3, author: '@A8FGVdYR8q2yKeluGzE2+R0CTdFQXtzDa+jHRWE0SAqB.secp256k1', timestamp: 1549395437481, hash: 'sha256', content: { type: 'post', text: 'its nice to be important but its more important to be nice' }, signature: 'dg6qh+kvti0jCASiNRlF/iPO+oWsDjrsAmuTD2ZJtENlfJ3RH2bf5xbetF/kbsO+N3Al4YbCIohextle4bAa3w==.sig.secp256k1' }, timestamp: 1549395437491 }

Security issues

Allowing these signatures on the Scuttlebutt network could introduce security issues, but at this proof-of-concept level we have not made this our primary concern. It is anyway worth noting the following:

The library we have used, 'secp256k1-node', explicitly states in the readme that it is an experimental module.

To protect against the 'chosen protocol attack', we don't want a valid SSB message to also be a valid Etherum transaction ever

. In SSB this is addressed by using the hmac_key

argument to ssb-keys.signObj

. The idea is that there is a separate hmac_key

for each type of thing which one is able to sign, and then signObj(keys, hmac_key, obj)

is the same as sign(keys, hmac(obj, hmac_key))

. This means it is almost impossible that hmac(obj, hmac_key)

could also be a valid Ethereum transaction.

Scaling issues

SSB is an agent-centred protocol where data is replicated based on relationships between agents. As already mentioned, in contrast to DHT-based peer-to-peer protocols, there is no one ubiquitous view of the network, but each node has access to some limited area of the social graph. This means it scales very well, but only when the application is relationship-centred in nature.

Use in State Channels

Due to it's agent-centred or relationship-centred design, SSB has implications for state channels, and state channel networks. In Counterfactual's paper on 'Generalised State Channels' it is made clear that with state channels, users hold more data than simply their public key, and storage and transport of this data is an issue for application developers. The automated replication of SSB feeds to neighbors on the trust graph gives us a good compromise between privacy and data persistence. Tampering of a feed by peers is made impossible by using timestamps, hash-linking and signatures.

Similar/complementary efforts

ConsenSys have implemented SSB in Java as part of their core library, 'Cava'. The idea is to make it interoperable with RLPx (the protocol with which Ethereum nodes communicate with each other).

Possible alternative approaches

Just having a 'this is my other key' message

Using the same keypair to sign SSB messages as well as blockchain transactions allows us to reliably refer to the same key holder on both systems. But in the context of the current uses of the Ethereum and Scuttlebutt networks, there is also a disadvantage. The security with which you safeguard a key to a wallet or smart contract is fundamentally different to the security required to safeguard a key for signing casual messages, and re-using keys for multiple purposes is often regarded as bad practice. Using the same keypair for both would compromise security and/or introduce impractical conditions for sending SSB messages (having to enter passwords or requiring hardware devices). A better option could be a way to look up and verify the associated addresses on both platforms.

We could publish a public SSB message, of a particular message type so that it can easily be found, with a secp256k1 public key or Ethereum address. This message could included signatures made with both the Ethereum key and the SSB key. This would make it easy to, say, send funds via Ethereum to someone with a known SSB identity. Importantly, this could be done entirely in 'message space' with no protocol changes needed. Similarly, a message announcing an SSB feed id could be published to the Ethereum blockchain to allow the inverse - finding an SSB identity associated with a known SSB address.

Multiple feeds using a single secure channel.

Another consideration is to have a single user or entity owning multiple SSB feeds, some which have different capabilities than others. Imagine I have a 'normal' ed25519 SSB identity, which I use to establish a secure channel with peers by means of SSB's secret-handshake. I also have as second identity with some special property, such as signing with a secp256k1 key. As the two feed-ids are 'friends', i am able to use the secure channel created by the first id, to replicate the second feed, meaning we don't have to worry about the handshake protocol working with both systems, or security issues with reusing keypairs for many purposes.

sameAs

sameAs

is an SSB message type used to create links between SSB accounts such that a single identity can comprise of multiple accounts. The primary use case is users with multiple devices, but it could be used to link accounts that have particular properties or abilities (such as signing on a different curve).

'blockparty'

'blockparty' is a multi-network SSB client which aims to create interoperability between different 'scuttleverses' (the 'main' scuttlebutt network has a particular network key, so there are many possible alternatives). Taking this project as inspiration, we could have an alternative network where secp256k1 feed ids are the norm, but have some cross-references to the 'main' network, meaning we can make use of existing accounts and infrastructure such as pubs.

Conclusion

We have implemented secp256k1 keys for signing, and allowing this on the main Scuttlebutt network is an obtainable goal. We have also identified some aspects of the SSB protocol which might be beneficial to Dapp developers. In particular, SSB's gossip protocol is suited to agent-centred, as opposed to content-centered contexts, which has implications for state-channels. It is also well suited for transport and persistence of sensitive information, where it is important that metadata is

obscured.