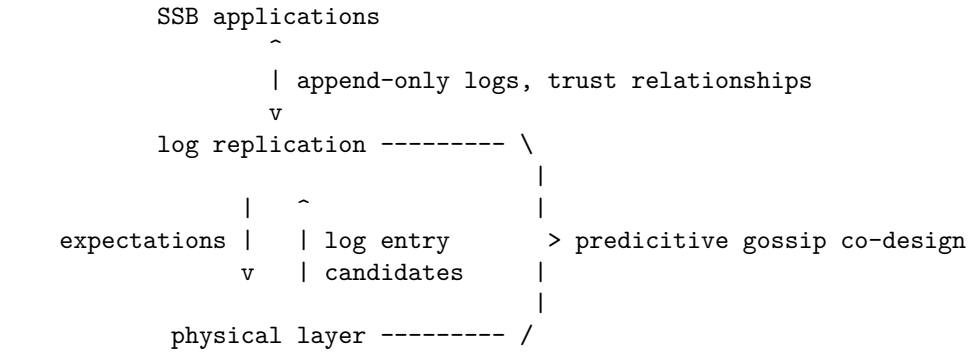
Predictive Gossip

2020-03-10 christian.tschudin@unibas.ch

(This document is obsolete wrt technical details but captures some of the rationale for tinySSB. See the packet-spec for more relevant information.)

Our goal is to define a minimal dissemination layer for a Secure Scuttlebutt-like network (SSB) that enables cross-layer optimizations suitable for challenged radio environments. Starting from the observation that SSB communication has considerable predictability, we expose that predictability to the decoding process of the physical layer. The physical layer decodes symbols with a correctness probability where expectations from higher layers are helpful *a priori* information for iterative decoding. Because of the predictability of SSB communications, higher-level expectations can be made explicit. For example, only feeds we subscribed to (=followed) will be considered (are thus predicted), and the set of acceptable sequence numbers is very small and also predictable. All predictable data can be omitted from transmission, saving precious bandwidth. In fact, our packet format has no fields for feedID, sequence number or prevMesgID which are mandatory in full SSB! See also the companion "Rumpelstiltskin Talk" paper that shows that even non-predictable data can be ommited to some extend.

The following diagram shows the three layers of our architecture:



Ed25519 identities are used at two levels: (a) applications where users interact via global append-only logs, (b) replicating devices using multiple link-local append-only logs for *managing* replication of the upper-level logs. In terms of classic SSB, pubs correspond to our replicating devices. Unlike SSB, we assign independend IDs also to the used smartphones and laptops.

```
global logs: userX, metafeeds  <---social/follow relations---> userY, metafeeds

local logs:     device1 <---topological/mgmt relationships---> deviceN
```

Assumptions and Goals

- physical/logical, local/regional/global broadcast setting: need to demux the many sources
- resource-constrained environment: lean append-only logs with very small entries
- datagram-based native SHS instead of TCP-based SHS
- payload, despite being small (48B), sufficiently large to do Signal's x3dh handshake or SHS
- peer-to-peer, must support symmetric replication
- only supports hop-by-hop interaction (for replication, and for replication management)
- we accept content only from known sources: unexpected content is discarded as if corrupt
- zero configuration except trust relations
- metadata-hiding by default (cloaking), based on exploiting predictability/compression

. . . this section to be sorted, expanded, rewritten

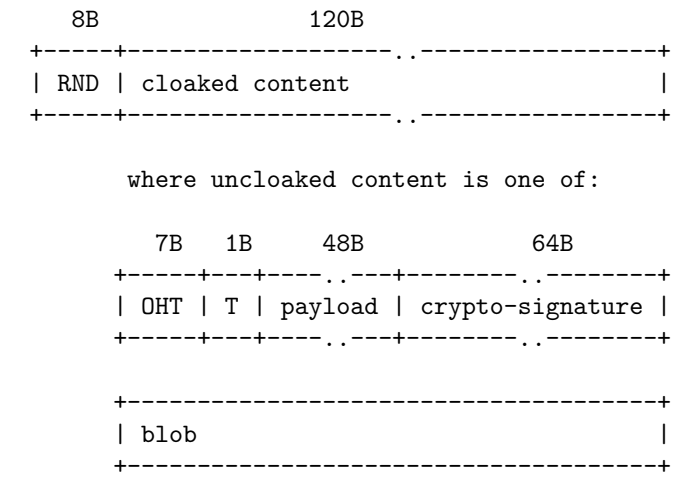
A Small Hybrid Packet Format for Replication of Append-only Logs

hybrid = predictive content transfer *and* traditional header fields

small = suitable for transmission over amateur radio channels (3kHz or less)

General packet layout

Each packet is exactly 128 bytes long:



The following table summarizes the various fields:

field	comment
RND	key for a bit mask that is xored with the content
OHT	Oblivious Hash Tag: serves as filter for feedID and packet purpose
T	specifies the used signing algo and payload type
payload	body of an append-only log entry or a replication mgmt cmd
signature	e.g., ed25519, computer over a virtual data structure
blob	any content

Cloaking: A packet's outer layer cloaks the 120 content bytes: the randomly picked RND field, together with the ID (typically a public key) of the sending *device*, is used to compute a keyed hash value that is xored to the packet's content. The effect is that only an observer knowing the source of a packet is able to access the uncloaked content (we assume a broadcast medium like radio). This provides (radio) source anonymity unless the sender's location can be determined by triangulation. Moreover, when packets are retransmitted due to ARQ, cloaking make each transmission to look different with very high probability, preventing most traffic analysis.

The default configuration is that SSB relay nodes (physical devices) possess a key pair. Each device maintains trust relationship with a list of buddy devices for which it knows their public key. When receiving a packet, the RND value is tried with each public key from this list to see whether uncloaking yields a "useful" packet (see later what "useful" means). Note that the RND field is not part of the signed data and changes from hop to hop as a log entry is replicated across the network.

OHT: The "Oblivious Hash Tag" helps a receiving node to deciding quickly whether a given packet should be checked for a valid signature (which is computationally heavy), and/or for recovering implicit

content. The OHT serves as one of the “usefullness filters” mentioned in the previous paragraph: if the OHT value is not in the table of expected OHT value, it is not useful to process it further and the cloaking layer may try to uncloak the packet with another sender ID.

The OHT is typically formed by hashing the expected log entry name, which is the log owner (feed ID) and the sequence number. The OHT is obtained by truncating above hash, thus only provides a probability signal that the packet contains a legit payload. Note that the use of OHT means that a node must have an extensive list of hash values regarding all packets that are currently expected. This covers all possible log extensions but potentially also the positions just before and after, in order to give useful feedback regarding duplicate or lost packets. A second set of “pre-armed” hash values regards the management dialogue between two nodes.

Signature: The ultimate filter deciding on the acceptance of a packet is based on the signature field. This signature is computed over a virtual packet (that is never transmitted) that contains for example the source ID, the sequence number, the message ID of the previuous log entry, the T and the payload fields, and potentially some additional bits that are transferred using “Rumpelstiltsik Talk” (see the companion document). The filtering step based on the signature guarantees that only expected packets are processed, in this case are appended to a log’s replica.

T and **payload**: T contains 8 bits for specifying the signing algorithm, the log format, as well as the payload’s type. One type is *inline* meaning that the 48B payload bytes form this log entry’s body. Another type is *chained-content* where the payload contains 28 bytes for a total length field, some content and a RIPEMD-160 hash value that identifies a manifest packet. A manifest packet is a 120 bytes data structure with 100 content bytes followed by another RIPEMD-160 pointer. This permits to have arbitrary long log entries. Other type (for further study) include *binary_tree_content*.

manifest packets and **blobs**: 120 bytes-long data can be shipped with the same cloaking mechanics as the structured packets described above. While the structured packets also have a OHT field, blobs occupy the full 120 bytes available. The decision about “usefulness” is again done based on expectations: only blobs asked for will be considered. For example, when a node receives a log entry with a *chained-content* type, it will ask its peer to start streaming the manifest chain, and add the RIPEMD-160 value of the first manifest to its acceptance-by-hash table. After reception it will replace this entry by the RIPEMD-160 found in the first manifest etc, until the whole chain has been received. The same filtering applies for blobs created at application layer: As in full SSB, an application can request to receive a blob (part of a chain or tree), which translates into the replication module issueing that request to its buddies and arming its hash filter for the incoming blob fragments. A blob fragment can be a manifest (chain or tree node) as well as a leaf node (pure data), which is a distinction that is known at query time and thus also can be turned into an expectation. Note that blobs do not need a signature because they are referenced in some authenticated log entry, thus inherit this property even if part of a tree or chain.

Computing the various packet fields

this section also shows different use cases for the packet format, beyond sending log entries.

| stands for concatenation

1) input for sending a log entry: feed ID, seqno, prevMsgId, algo, payload

```
OHT        = truncate( hash('SSB' + feedID + enc(seqno)) )
T          = algo and other packet type details
virtPacket = feedID | enc(seqno) | T | prevMsgId | payload
signature  = ed25519sign(feedID.sk, virtPacket)

content    = OHT | T | payload | signature
```

```
rnd = random
pad = hash(rnd, feedID) // optionally: add a secret per-link prefix
                        // to feedID or RND for making it unfeasible to spot
                        // specific feedID packets by exhaustively searching
                        // the RND space

--> output: rnd | (content XOR pad)
```

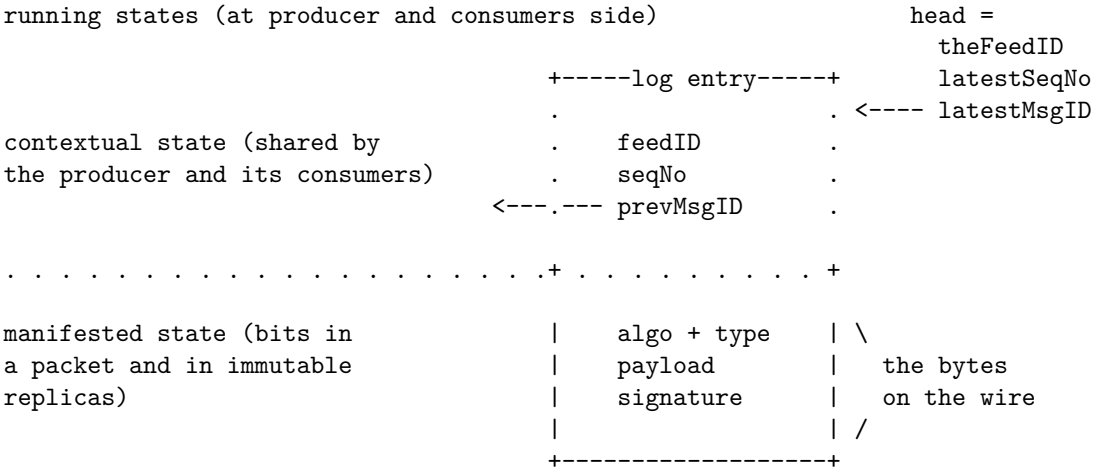
Note that from the four inputs, only algo and payload are encoded. The other three fields are fully predictable by the receiver and will be inserted in a "virtual packet" before verifying the packet's signature.

2) Packet content for device-to-device connection setup (Signal's X3DH protocol)

```
a) Announcing prekeys (Bob)
   DMX = trunc( hash('X3DH' + BobID) ) // buddies are able to filter on this
   1B T = algo
   48B payload = prekey and nonce N
   64B signed by Bob

b) Initiate1 (Alice)
   DMX = trunc( hash('X3DH-SYN' + BobID + N) )
   1B T = algo
   112B:
      . 48B encr(msg16, AAD=BobID+N), (16B nonce + 32B msg)
      . 32B AliceID
      . 32B ephemeralKey
   where msg is feedID of alice-to-bob log
   and encr() is encrypted with the shared DH-secret

c) Initiate2 (Bob's answer)
   OHT = trunc( hash('X3DH-SYNACK' + BobID + N ) )
   1B T = algo
   112B:
      . encr(msg16, AD=Alice_ID|Bob_ID), (16B nonce + 48B)
      . ephemeralKey (32B)
      . AliceID (32B)
   payload contains new "bob-to-alice feed ID",
   and encr() is encrypted with the shared DH-secret

d) Initiate3 (Alice's answer)
   OHT = trunc( hash('X3DH-ACK' + N + AliceID) )
   payload contains new "alice-to-bob feed ID"
   encrypted with the shared DH-secret

From now on, the two devices Alice and Bob run their
replication management protool in these logs
(corresponds to the stream of RPC msgs in SHS)
```

Visualizing the Hash Chain

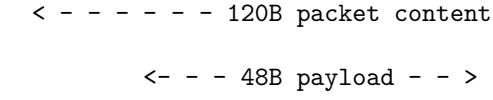
Each append-only log is implemented, as in SSB, as a hash chain where a log entry always points to its predecessor entry via a “hash pointer” (prevMsgID). Unlike SSB, however, not all fields of a log entry are materialized when being transmitted. For example, the sequence number is an implicit property that can be computed by counting the number of entries that preceed a given log entry (the first log entry has sequence number 0). Other properties are also contextual, meaning that a subscriber to a log will be able to collect and update these properties while consuming the log: The feedID is a constant contextual property that can be factored out and will not be part of the manifested bytes of a log entry. The latestMsgId is a variable contextual property that is updated each time the log gets an extension and belongs tp the head state that each party keeps to characterize the hash chain. The following pictures shows the two classes of fields, those which are contextual (thus not encoded at packet level) and those which are non-predictable (thus are manifested at packet level), as well as (in-memory) head state:

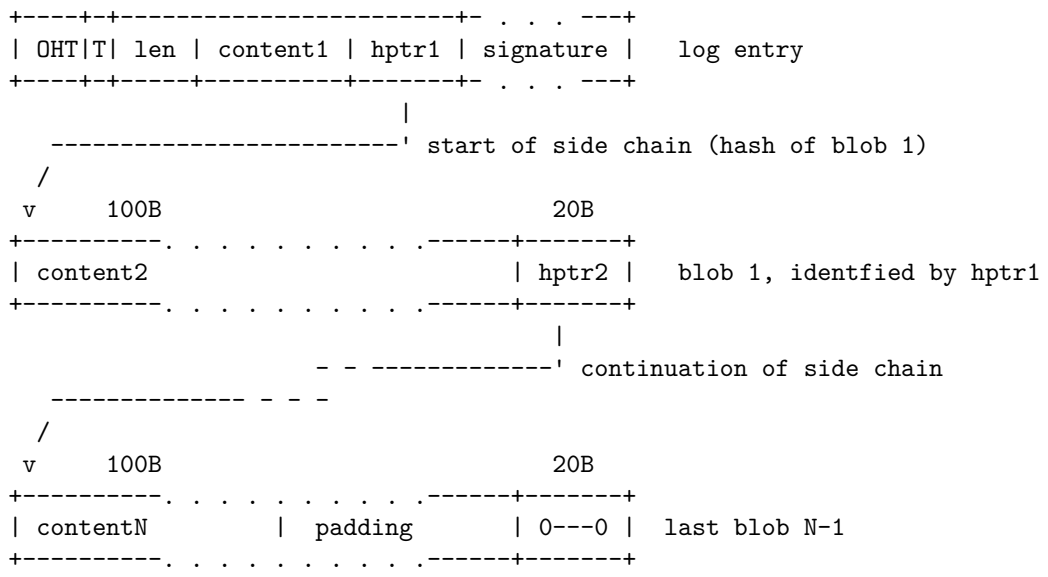


Although not all fields of a log entry are materialized when transmitted, the signature shall nevertheless cover them all. That is, the producer computes the signature over a full “virtual log entry” and consumers recreate that virtual log entry before verifying the received signature.

Another difference to classic SSB is that the payload of a log entry can be arbitrary long by using a side-chain construction. That is, a “log entry with extended payload” consists of at least one signed packet plus an arbitrary number of unsigned chained “blobs”, all having the same total length of 128 bytes. The payload in the log entry packet contains the over length of the “real payload” content, as well as a hash value that points to the first extension blob. Blobs reserve the last 20 Bytes for chaining where a zero-value means that the side chain ends here. The RIPEMD-160 hash function is used due its small hash value size of 20 bytes.

Whether a log entry has a plain payload (not using the extension) or whether its content is spread over a side chain, is encoded in the log entry’s type field T. When replicating, neighbor devices can choose whether they want to receive only the log entries, or whether they want the stream of packets to also include all associated side chain blobs. The side chains can be re-requested later with a similar choice between receiving only one blob or the sequence of chained blobs. Note that in all cases the receiving device can predict which blob will come next and hence can arm its “predicted filter bank” to have an entry for the next chained blob to be accepted (called “usefulness” in a previous section).





Note that when forming the chain, the last blob has to be created first. The length (of the concatenated content fields) is encoded in a variable-length format e.g, Bitcoin's varInt, and its length must be considered when filling the last packet.

---

more to come ...

Appendix

A) Considerations for packet header lengths

- 2**N overall and constant length of each packet, for optimal packing
- payload extensibility: note that blob chains or tree can be arbitrary large

128 = 8 + 7 + 1 + 48 + 64

Overall length:
For the faintest useful radio channels, sending 1000 bits with 1kHz bandwidth might need 10 sec already. Increasing packet size would move us outside reasonable delays or reduce distance below intercontinental contacts.

Are RND and OHT balanced? OHT is necessary for DoS mitigation, preventing unnecessary signature verifications which are slow. Hashed values are also good for loadbalancing with sharding, but 7 bytes for OHT seems a lot?

Note that the cloaking layer (RND) is a wrapper and two nodes may decide to operate without it. Adding this space, together with a smaller OHT, we could have 128 = 0 + 3 + 1 + 60 + 64 (those environments which want cloaking would have to send 6+128 bytes - there is no need

```
                                to store the RND value which was picked
                                randomly and carries no information, hence
                                packing would still be multiples of 128).

                                But 2**24 for OHT is not giving much protection
                                in high speed networks: Assume packed Eth
                                frames with 10 packets each and at 1 Gpps,
                                this space wraps around in 1.5 millisec.
                                If such a node had a modest 10'000 filter rules,
                                a sweep attack leads to 6.5 Mio forced signature
                                verifications on top of valid packets.

                                With 5 OHT bytes, this becomes 100 seconds,
                                or 100 "filter hits" per second triggering
                                a signature verification, which is tolerable.
                                I.e., OHT should be at least 5B wide.

-->                             The format 128 = 6 + 5 + 1 + 52 + 64 would
                                have the nice property that the 52B payload
                                gives space for a RIPEMD-160 hash and a
                                32B field at the same time (e.g., an ed25519
                                public key). But do we have a use case for this?

                                Similar thoughts for 128 = 0 + 5 + 1 + 58 + 64
                                where RND is outside of the equation and could
                                be made arbitrarily large depending on the
                                link? This would permit to harden cloaking
                                where this is critical. On the other hand,
                                moving RND outside the 128 bytes creates a
                                gradient for dropping cloaking by default.
                                Cloaking traffic for an Internet overlay
                                is less useful, but could be attractive for
                                implementing rooms (onion routing-like tunnels?)

 256 = 8 + 7 + 1 + 176 + 64     overall size wastefull, even prohibitive
                                for small sensor networks?
```

B) Considerations regarding the hash load induced by repeated trial-and-error parsing

When receiveing a packet, the RND-based uncloaking has to be tried for every buddy from which the receiving device expects a packet. To decide whether the right buddy was guessed, the OHT filter must be looked up in the table of expected packet tags as well as in the table of expected blobs, for which 120 Bytes have to be hashed for each “RND candidate”. Overall, every packet will trigger several hash computations, both at the level of packet fields (RND-based derivation of the cloaking bits), at the packet level (compute the uncloaked content’s hash for finding expected blobs), and finally also inside the SW: lookup in tables will use (non-cryptographic) hashing.

Weakening the RND and OHT hashing to non-cryptographic algorithms is not advised, especially not in the latter case where we need to protect the “hidden names” input to the hash functions (see the companion “Rumpelstiltskin Talk” paper).

Trial-and-error at cloaking level can be optimized to first try the sender IDs of peers from which one has recently received packets and probing for “almost fogotten buddies” only if this fails. In case a device

that operates in an Internet overlay, only buddies with which an active TCP connection exist, would be considered for uncloaking.

We think that modern CPUs have considerable “hash power” and that highspeed replicators can be built that use GPU-based hash accelerators and parallel search. For small devices, however, hashing requirements will lead to high processing times (when compared to plain IP store-and-forward) and might prevent some real-time applications if they have many buddies. Dropping the cloaking layer reduces the load drastically if a device has many buddies, hence offers one possible tradeoff. Alternatively, link-local encryption could be installed at the cost of a stateful security protocol for creating and operating a secure tunnel and revealing a packet’s uncloaked content with one decrypting step. However, aiming at a simple replication layer, we wish to avoid the definition of a link-local encryption protocol or mandating TLS.

C) Expectation Expectations (mental reference points)

Dialogs among humans work because of their predictability and “expectation expectations” (Luhmann): Not everything can be said at any time, only topic categories that the other side expects from me to expect from them will be sent to me, because they know that any other news would puzzle me. As a consequence, we have some *a priori* knowledge about what is going to be said. This reduces communication complexity as otherwise one would have to laborously zoom in to the context in which some news belongs.

Coming from the other end, namely physical signals, *a posteriori* knowledge is used in modern coding theory that optimizes the way of choosing the most probably signal decodings among many alternatives, by iteratively narrowing down the set of possibly received symbols.

Recent communication protocols like FT4 and FT8 from radio amateurs apply both ‘a priori’ (top-down) and ‘a posteriori’ (bottom-up) decisions to extract digital messages from very faint signals. The common element of both decoding strategies is that they act as *filters* where improbable or unexpected content is discarded, either in a pipeline or an iterative process.

In this work we deliberately let a sender compose `digital messages that are incomplete by design`: all items that the receiving peers *could* know in advance will be omitted. This results in minimal packet content where the receiver now *must* know the missing pieces in order to correctly parse the received messages. In signaling theory this compression is naturally called source coding and is used e.g. in the UDP header compression protocol. Beyond compression, shared (and moving) state is the basis of the double-ratchet crypto protocol where peers can only decipher their messages when they are in possession of a constantly and jointly changing secret – with the purpose to provide forward secrecy. This last example can be seen as a materialization of the “expectation expectations” principle introduced above.

`eof`