A Cryptographic Investigation of Secure Scuttlebutt

Oleksandra Lapiha

DIENS, École normale supérieure, Paris, France

oleksandra.lapiha@ens.fr

2 September 2019

What is Secure Scuttlebutt?

decentralized secure gossip platform

- No central authority controls the network.
- Ambitious security goals!
- Peer-to-peer communication is based on the "epidemic spread".
- Framework for building applications.

Core Concepts

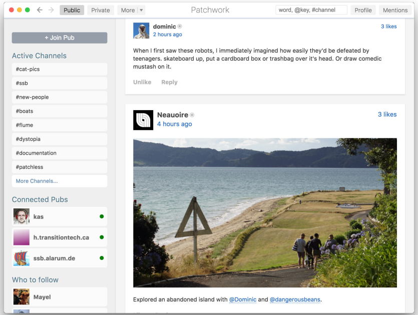
- *"Scuttlebutt Client"*: a member of a Scuttlebutt network.
- *"Message feed"*: A sequence of blocks of information chained by a hash function.
- *"Pub"*: Scuttlebutt client which is run on the server and assumed to be always online.

Scuttlebutt's Functionality

- Synchronizing message feeds over a local network.
- Pubs as a way to synchronize with remote peers.
- Following message feeds.
- Private group chats for 2 to 7 participants.
- Private Scuttlebutt networks.
- Operating fully offline.

Applications

- Social network: patchwork



dominic
2 hours ago
3 likes

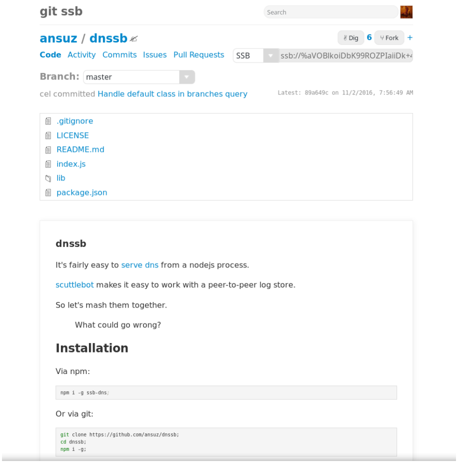
When I first saw these robots, I immediately imagined how easily they'd be defeated by teenagers. skateboard up, put a cardboard box or trashbag over it's head. Or draw comedic mustash on it.

Neauoire
4 hours ago
3 likes

Explored an abandoned island with @Dominic and @dangerousbeans.

Applications

- Social network: patchwork
- Source code repository: git-ssb



git ssb

ansuz / dnssb

ssb://%aVOBlkoiDbK99ROZPIaiiDk+4

Branch: master

cel committed Handle default class in branches query

Latest: 89a649c on 11/2/2016, 7:56:49 AM

- .gitignore
- LICENSE
- README.md
- index.js
- lib
- package.json

dnssb

It's fairly easy to serve dns from a nodejs process.

scuttlebot makes it easy to work with a peer-to-peer log store.

So let's mash them together.

> What could go wrong?

Installation

Via npm:

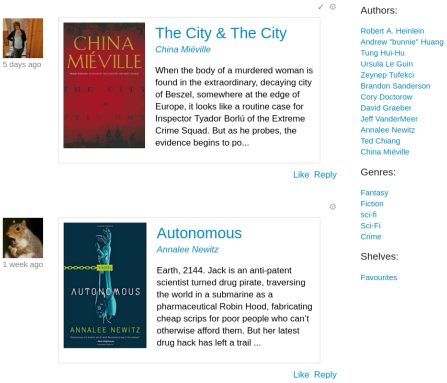
```
npm i -g ssb-dns;
```

Or via git:

```
git clone https://github.com/ansuz/dnssb;
cd dnssb;
npm i -g;
```

Applications

- Social network: patchwork
- Source code repository: git-ssb
- Book sharing: patch-book



5 days ago

The City & The City

China Miéville

When the body of a murdered woman is found in the extraordinary, decaying city of Beszel, somewhere at the edge of Europe, it looks like a routine case for Inspector Tyador Borlú of the Extreme Crime Squad. But as he probes, the evidence begins to po...

Like Reply

1 week ago

Autonomous

Annalee Newitz

Earth, 2144. Jack is an anti-patent scientist turned drug pirate, traversing the world in a submarine as a pharmaceutical Robin Hood, fabricating cheap scrips for poor people who can't otherwise afford them. But her latest drug hack has left a trail ...

Like Reply

Authors:

Robert A. Heinlein
Andrew "bunnie" Huang
Tung Hui-Hu
Ursula Le Guin
Zeynep Tufekci
Brandon Sanderson
Cory Doctorow
David Graeber
Jeff VanderMeer
Annalee Newitz
Ted Chiang
China Miéville

Genres:

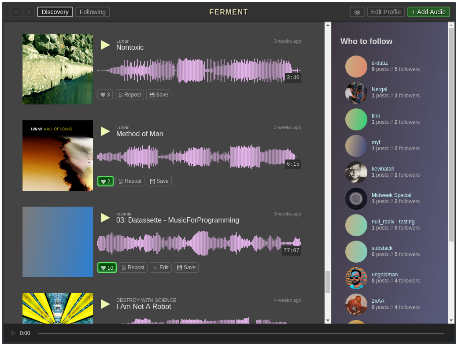
Fantasy
Fiction
sci-fi
Sci-Fi
Crime

Shelves:

Favourites

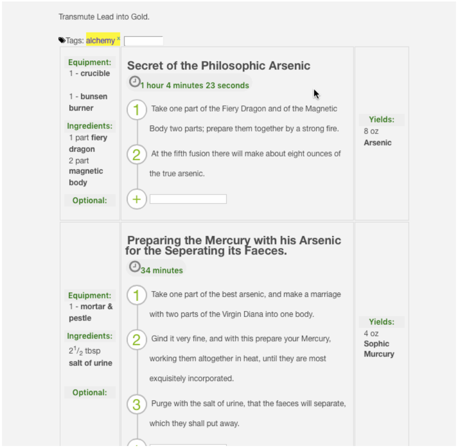
Applications

- Social network: patchwork
- Source code repository: git-ssb
- Book sharing: patch-book
- Music sharing: Ferment



Applications

- Social network: patchwork
- Source code repository: git-ssb
- Book sharing: patch-book
- Music sharing: Ferment
- Recipe sharing: ssb-recipes


Transmute Lead into Gold.

Tags: alchemy ×

Equipment: 1 - crucible
1 - bunsen burner
Ingredients: 1 part fiery dragon
2 part magnetic body
Optional:

Secret of the Philosophic Arsenic
1 hour 4 minutes 23 seconds
1 Take one part of the Fiery Dragon and of the Magnetic Body two parts; prepare them together by a strong fire.
2 At the fifth fusion there will make about eight ounces of the true arsenic.

Yields: 8 oz Arsenic

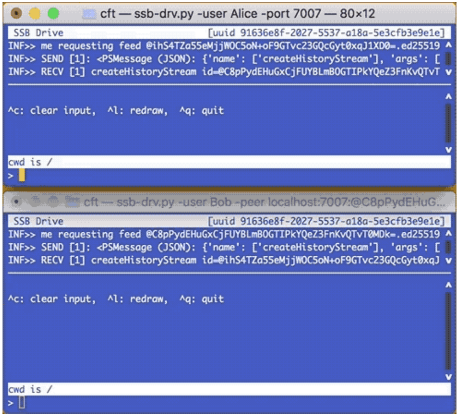
Equipment: 1 - mortar & pestle
Ingredients: $2^1/_2$ tbsp salt of urine
Optional:

Preparing the Mercury with his Arsenic for the Seperating its Faeces.
34 minutes
1 Take one part of the best arsenic, and make a marriage with two parts of the Virgin Diana into one body.
2 Gind it very fine, and with this prepare your Mercury, working them altogether in heat, until they are most exquisitely incorporated.
3 Purge with the salt of urine, that the faeces will separate, which they shall put away.

Yields: 4 oz Sophic Murcury

Applications

- Social network: patchwork
- Source code repository: git-ssb
- Book sharing: patch-book
- Music sharing: Ferment
- Recipe sharing: ssb-recipes
- File system: ssbdrv


cft — ssb-drv.py -user Alice -port 7007 — 80×12

```
 SSB Drive                            [uuid 91636e8f-2027-5537-a18a-5e3cfb3e9e1e]
INF>> me requesting feed @ihS4TZa55eMjjWOC5oN+oF9GTvc23GQcGyt0xqJ1XD0=.ed25519
INF>> SEND [1]: <PSMessage (JSON): {'name': ['createHistoryStream'], 'args': [
INF>> RECV [1] createHistoryStream id=@C8pPydEHuGxCjFUYBLmBOGTIPkYQeZ3FnKvQTvT

^c: clear input,  ^l: redraw,  ^q: quit

cwd is /
>
```

cft — ssb-drv.py -user Bob -peer localhost:7007:@C8pPydEHuG...

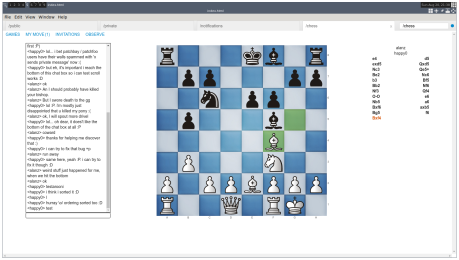
```
 SSB Drive                            [uuid 91636e8f-2027-5537-a18a-5e3cfb3e9e1e]
INF>> me requesting feed @C8pPydEHuGxCjFUYBLmBOGTIPkYQeZ3FnKvQTvT0MDk=.ed25519
INF>> SEND [1]: <PSMessage (JSON): {'name': ['createHistoryStream'], 'args': [
INF>> RECV [1] createHistoryStream id=@ihS4TZa55eMjjWOC5oN+oF9GTvc23GQcGyt0xqJ

^c: clear input,  ^l: redraw,  ^q: quit

cwd is /
>
```

Applications

- Social network: patchwork
- Source code repository: git-ssb
- Book sharing: patch-book
- Music sharing: Ferment
- Recipe sharing: ssb-recipes
- File system: ssbdrv
- Chess: ssb-chess



first :P)
<happy0> lol... i bet patchbay / patchfoo users have their walls spammed with 'x sends private message' now :(
<happy0> but eh, it's important i reach the bottom of this chat box so i can test scroll works :D
<alanz> ok
<alanz> An I should probably have killed your bishop.
<alanz> But I swore death to the gg
<happy0> lol :P. i'm mostly just disappointed that u killed my pony :(
<alanz> ok, I will spout more drivel
<happy0> lol... oh dear, it does't like the bottom of the chat box at all :P
<alanz> coward
<happy0> thanks for helping me discover that :)
<happy0> i can try to fix that bug =p
<alanz> run away
<happy0> same here, yeah :P. i can try to fix it though :D
<alanz> weird stuff just happened for me, when we hit the bottom
<alanz> ok
<happy0> testarooni
<happy0> i think i sorted it :D
<happy0> l
<happy0> hurray \o/ ordering sorted too :D
<happy0> test

alanz	happy0
e4	d5
exd5	Qxd5
Nc3	Qe5+
Be2	Nc6
b3	Bf5
Bb2	Nf6
Nf3	Qf4
O-O	e6
Nb5	a6
Bxf6	axb5
Bg5	f6
Bxf4	

Security Goals

- *Mutual authentication:* The client must know the server's public key before connecting. If the handshake is successful peers' public keys are verified.
- *Identity and Network identifier hiding:* A man-in-the-middle cannot learn the public key of either peer or the network id.
- *Network confirmation:* If the handshake succeeds then both ends have confirmed that they wish to use the same network.
- *Replay attack resistance:* Attempting to replay a handshake will not allow an attacker to discover the participants' public keys.
- *Forward secrecy:* Recording a user's network traffic and then later stealing their secret key will not allow an attacker to decrypt their past handshakes.

Protocol Structure

- *Handshake:* initialize the synchronization with authenticated key exchange.
- *Invite Redemption:* join the Pub by redeeming the invite code.
- *Private Messaging:* broadcast encrypted messages for 1-6 other participants.

Related Work

Cremers, Cas and Jackson, Dennis (2019)
Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman.

- Analyze Scuttlebutt Handshake Protocol.
- Model the protocol in Tamarin.
- Find a Small subgroup attack and propose a fix.

Contributions

- Analyze Handshake, Invite Redemption and Private Messaging protocols.
- Model the protocols in ProVerif.
- Confirm the results of Cremers and Jackson by detecting the Small subgroup attack.
- Model the handshake and invite redemption protocols in CryptoVerif with game-based proof for certain authentication properties.

Background: Symbolic Model

In the *symbolic verification model* the cryptographic primitives are represented by function symbols considered as *perfect black-boxes*, the messages are abstract terms, and the adversary is restricted to compute only the primitives defined by the user.

$$\begin{array}{l}\text{fun enc(bitstring, key) : bitstring.}\\ \text{fun dec(bitstring, key) : bitstring}\\ \text{reduc forall } m : \text{ bitstring, } k : \text{ key;}\\ \text{dec(enc(}m, k), k) = m\end{array}$$

Background: ProVerif

- *ProVerif* - fully automated symbolic protocol verifier.
- Proving strategy: search for a protocol trace that violates the security goal.
- Developed by *Bruno Blanchet and Vincent Cheval* in Inria, Paris.
- Protocols are described in applied pi-calculus.
- The structure of the model: $\Sigma = \Delta_1 \ldots \Delta_n P$
- Where Δ_i is a declaration of the : *type, free name, query, constructor, destructor, equation, pure function or a process*
- And P is a top level process.

Primitives: Modeling Unexpected Behavior

fun get_pk(sk) : pk.
fun sign(bitstring, sk) : bitstring.
fun checksign(bitstring, pk, bitstring) : bool
reduc forall m : bitstring, k : sk;
checksign(sign(m, k), get_pk(k), m) $=$ true

Primitives: Modeling Unexpected Behavior

fun get_pk(sk) : pk.
fun weak(sk) : sk.

fun sign(bitstring, sk) : bitstring.
fun checksign(bitstring, pk, bitstring) : bool
reduc forall m : bitstring, k : sk;
checksign(sign(m, k), get_pk(k), m) = true
otherwise forall $m1$: bitstring, $m2$: bitstring, k : sk;
checksign(sign($m1$, weak(k)), get_pk(weak(k)), $m2$) = true

Primitives: Modeling Unexpected Behavior

fun get_pk(sk) : pk.

fun exp(pk, sk) : sym_key.
equation forall x : sk, y : sk;
exp(get_pk(x), y) $=$ exp(get_pk(y), x).

Primitives: Modeling Unexpected Behavior

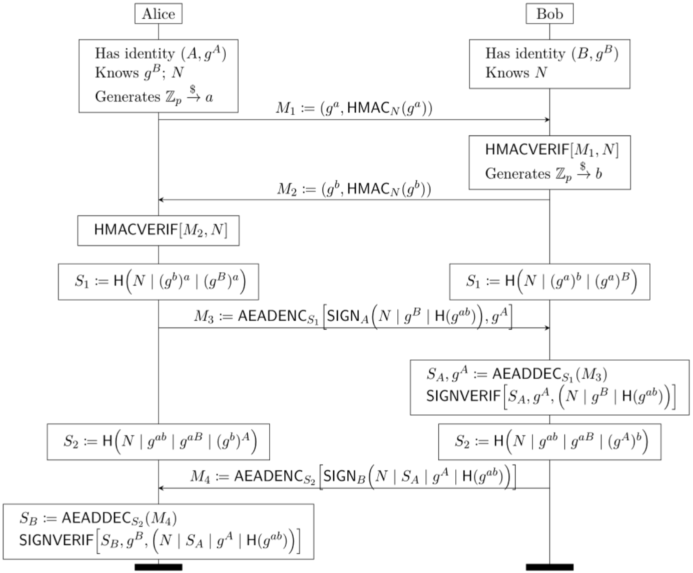
```
const zero : sym_key [data].

 fun get_pk(sk) :  pk.
 fun weak(sk) :  sk.

 fun exp(pk, sk) :  sym_key.
 equation forall x : sk,  y : sk;  exp(get_pk(x), y)  =  exp(get_pk(y), x).

 fun dhexp(pk, sk) :  sym_key
 reduc forall b : sk,  a : sk;  dhexp(get_pk(weak(a)), b)  =  zero
 otherwise forall b : sk, a : sk;  dhexp(get_pk(a), weak(b))  =  zero
 otherwise forall b : sk,  a : sk;
 dhexp(get_pk(a), b)  =  exp(get_pk(a), b).
```

Handshake


Alice
Bob
Has identity (A, g^A)
Knows g^B; N
Generates $\mathbb{Z}_p \xrightarrow{\$} a$
Has identity (B, g^B)
Knows N
$M_1 := (g^a, \mathsf{HMAC}_N(g^a))$
$\mathsf{HMACVERIF}[M_1, N]$
Generates $\mathbb{Z}_p \xrightarrow{\$} b$
$M_2 := (g^b, \mathsf{HMAC}_N(g^b))$
$\mathsf{HMACVERIF}[M_2, N]$
$S_1 := \mathsf{H}\big(N \mid (g^b)^a \mid (g^B)^a\big)$
$S_1 := \mathsf{H}\big(N \mid (g^a)^b \mid (g^a)^B\big)$
$M_3 := \mathsf{AEADENC}_{S_1}\big[\mathsf{SIGN}_A\big(N \mid g^B \mid \mathsf{H}(g^{ab})\big), g^A\big]$
$S_A, g^A := \mathsf{AEADDEC}_{S_1}(M_3)$
$\mathsf{SIGNVERIF}\big[S_A, g^A, \big(N \mid g^B \mid \mathsf{H}(g^{ab})\big)\big]$
$S_2 := \mathsf{H}\big(N \mid g^{ab} \mid g^{aB} \mid (g^b)^A\big)$
$S_2 := \mathsf{H}\big(N \mid g^{ab} \mid g^{aB} \mid (g^A)^b\big)$
$M_4 := \mathsf{AEADENC}_{S_2}\big[\mathsf{SIGN}_B\big(N \mid S_A \mid g^A \mid \mathsf{H}(g^{ab})\big)\big]$
$S_B := \mathsf{AEADDEC}_{S_2}(M_4)$
$\mathsf{SIGNVERIF}\big[S_B, g^B, \big(N \mid S_A \mid g^A \mid \mathsf{H}(g^{ab})\big)\big]$

Security Queries

1. Mutual authentication	☑
2. Shared secret agreement	☑
3. Identity hiding (and replay attack resistance)	☑
4. Network identifier hiding	☑
5. Network confirmation	☑
6. Forward secrecy	☑

Small Subgroup Attack

Here we consider network identifier N to be public.

- The attacker chooses his long-term and ephemeral key pairs to be weak.
- The attacker derives a shared secret with Bob which is now a constant value
- The attacker signs a random message with his key, the signature verifies against any message.
- The final derived shared secret is a constant, so it is know by the attacker, rendering it usable to encrypt messages to Bob.

Security Queries

1. Mutual authentication	✓
2. Shared secret agreement	✗
3. Identity hiding(& Replay attack resistance)	✓
4. Network identifier hiding	✗
5. Network confirmation	✓
6. Forward secrecy	✗

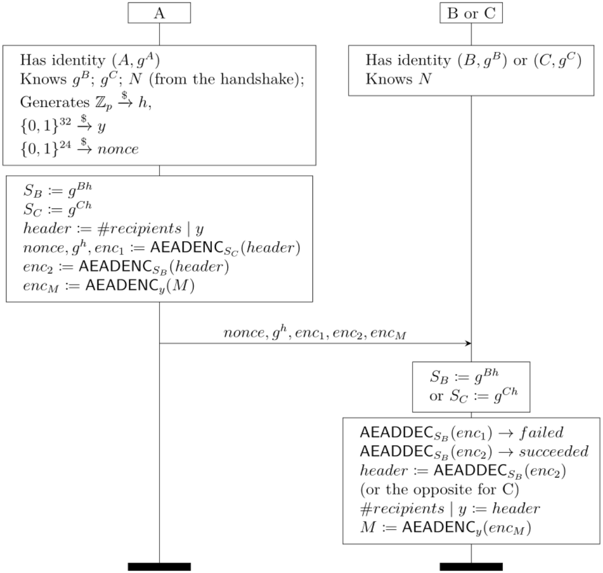
Invite Redemption

- Alice uses the invite key pair instead of her long-term key pair.
- Alice does not send any message to the Pub after the invitation code is used.

Security Queries

1. Mutual authentication	☒
2. Identity hiding(& Replay attack resistance)	☑
3. Network identifier hiding	☑
4. Network confirmation	☑

Private Messaging


A
B or C
Has identity (A, g^A)
Knows g^B; g^C; N (from the handshake);
Generates $\mathbb{Z}_p \xrightarrow{\$} h$,
$\{0,1\}^{32} \xrightarrow{\$} y$
$\{0,1\}^{24} \xrightarrow{\$} nonce$
Has identity (B, g^B) or (C, g^C)
Knows N
$S_B := g^{Bh}$
$S_C := g^{Ch}$
$header := \#recipients \mid y$
$nonce, g^h, enc_1 := \mathsf{AEADENC}_{S_C}(header)$
$enc_2 := \mathsf{AEADENC}_{S_B}(header)$
$enc_M := \mathsf{AEADENC}_y(M)$
$nonce, g^h, enc_1, enc_2, enc_M$
$S_B := g^{Bh}$
or $S_C := g^{Ch}$
$\mathsf{AEADDEC}_{S_B}(enc_1) \rightarrow failed$
$\mathsf{AEADDEC}_{S_B}(enc_2) \rightarrow succeeded$
$header := \mathsf{AEADDEC}_{S_B}(enc_2)$
(or the opposite for C)
$\#recipients \mid y := header$
$M := \mathsf{AEADENC}_y(enc_M)$

Private Messaging: Model

1. EncryptM42($skMe$: sk, $pkReceiver1$: pk, $pkReceiver2$: pk, m_out : bitstring)
2. DecryptM42($skMe$: sk, $pkSender$: pk)
3. EncryptM41($skMe$: sk, $pkReceiver1$: pk, m_out : bitstring)
4. DecryptM41($skMe$: sk, $pkSender$: pk)

Security Queries

1. Message authentication ☑
2. Forward secrecy ☑

Background: Computational Model

In the *computational model*, an adversary is a *probabilistic Turing machine*. It is restricted to run in polynomial time in security parameter. The security property holds if it can only occur with *negligible probability* in security parameter. We discuss the primitives with respect to underlying cryptographic assumptions.

```
expand IND_CPA_sym_enc(key, cleartext, ciphertext,
enc, dec, injbot, Z, Penc)
```

Background: Game-Based Proofs

- Given: a game between the adversary and a challenger...
- Goal: prove adversary's advantage is negligible.
- Game hopping: sequence of indistinguishable transformations of the initial game, such that for the final game the advantage is easy to upper bound.
- Transformations:
  - Transitions based on indistinguishability.
  - Transitions based on failure event.
  - Bridging steps.

CryptoVerif

- *CryptoVerif* - semi-automated computational protocol verifier.
- Proving strategy: generates a chain of indistinguishable games with aim to obtain a game where the property hold trivially.
- Developed by *Bruno Blanchet and David Cadé* in Inria, Paris.
- Protocols are described in applied pi-calculus.
- The syntax and the structure of protocol description is very close to ProVerif.

Results

- The Handshake

Theorem (Initiator Authentication and Network Confirmation)

If the initiator of the communication is able to complete the session it is authenticated to the responder. When parties complete the key exchange they agree on the network identifier.

- Invites

Theorem (Client Authentication and Network Confirmation)

If the client is able to complete the session, then his invitation key is accepted by the Pub. If parties were able to complete the protocol, then they agree on the network identifier.

Future Work

Assist the proof by CryptoVerif to:

- Prove more queries for existing models
- Model the private messaging protocol

Thank You!

- Work materials available at: `https://github.com/olapiha/scuttlebutt`
- *This research was performed as part of a summer research internship at Symbolic Software under thoughtful supervision of Nadim Kobeissi. :)*