HUNTING FOR VULNERABILITIES IN SIGNAL

JP Aumasson & Markus Vervier

WHOIS

- **JP** (@veorq)
  - Principal researcher @ Kudelski Security
  - Speaks French
  - Crypto guy
- **Markus** (@marver)
  - Head of research @ x41 D-Sec
  - Speaks German
  - Not CISSP

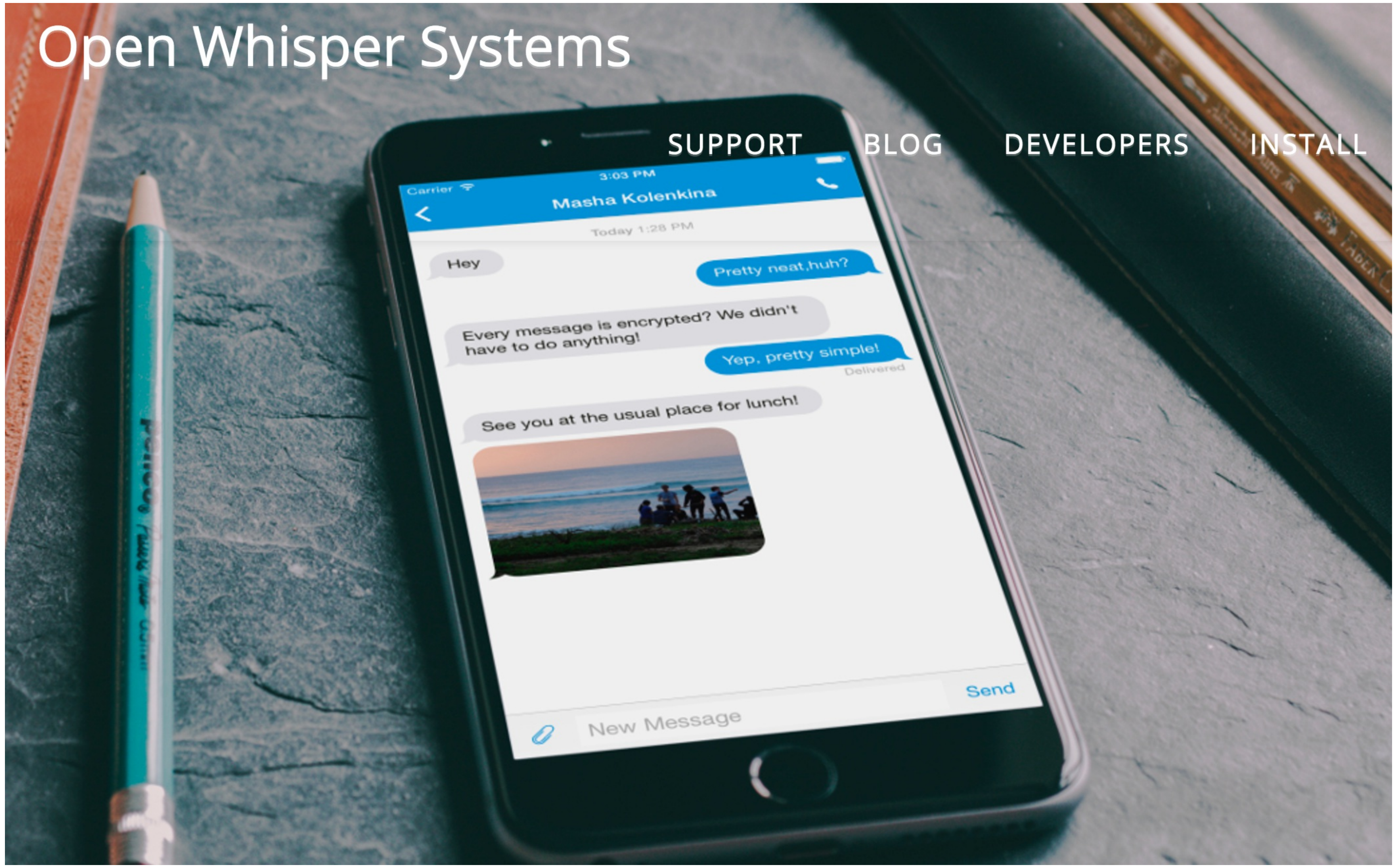
PROPS

- This BH US 2016 boring talk
- Open Whisper Systems
- Eric Sesterhenn
- Hanno Boeck

AGENDA

- **Signal** internals, security promises
- **Attack** surface and liabilities
- **Bugs**, **alternative features**, and demos
- Conclusions

SIGNAL



Today 1:28 PM

Hey

Pretty neat,huh?

Every message is encrypted? We didn't have to do anything!

Yep, pretty simple!

Delivered

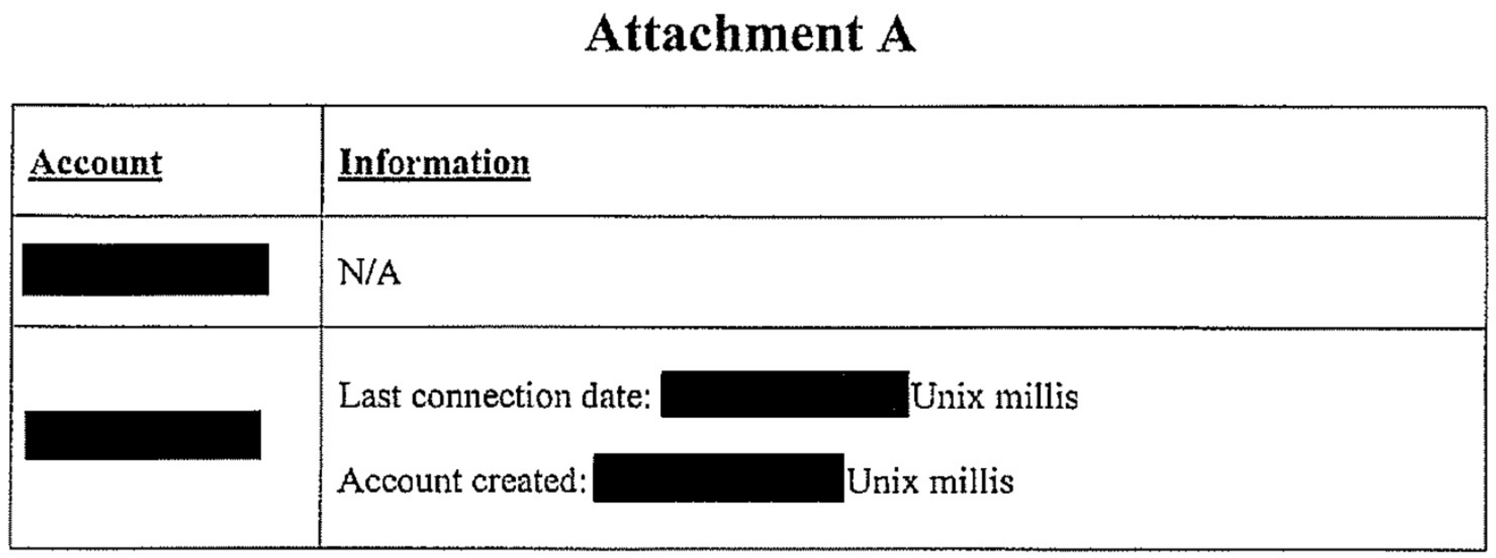
See you at the usual place for lunch!

THE SIGNAL APPS

- Mobile apps for **messaging** & audio/video **calls**
- By Open Whisper Systems (Moxie Marlinspike et al.)
- Formerly known as "TextSecure", "RedPhone"
- Android, iOS, and Chrome Desktop app

TRUSTED TOOL

- Endorsed by Snowden and other opinion leaders
- Popular among activists in the US and abroad
- Minimal data collection from Signal servers



Attachment A

Account	**Information**
[redacted]	N/A
[redacted]	Last connection date: [redacted] Unix millis Account created: [redacted] Unix millis

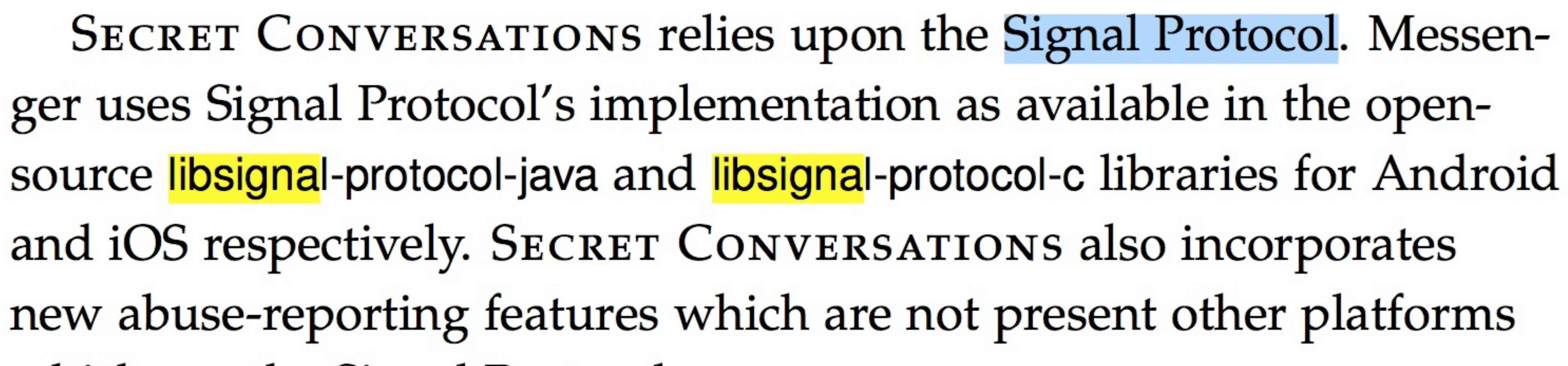
SECURITY PROMISES

- Solid **end-to-end encryption**, defending against
  - Active network attackers
  - Client and server compromises
  - Traffic analysis (partially)
- High assurance **software**, with
  - Code perceived as high-quality
  - No major security issue ever
  - Reproducible Android builds

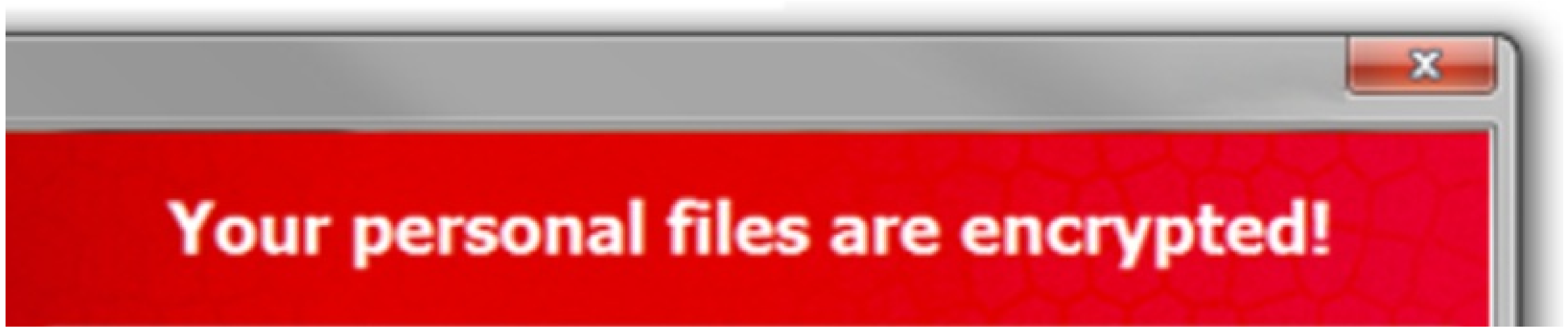
SIGNAL IS MORE THAN SIGNAL

Core crypto "libsignal" licensed to and integrated in

- Facebook **Messenger**'s "Secret Conversations" mode
- Facebook **WhatsApp** default encryption
- Google **Allo**'s "Incognito" mode

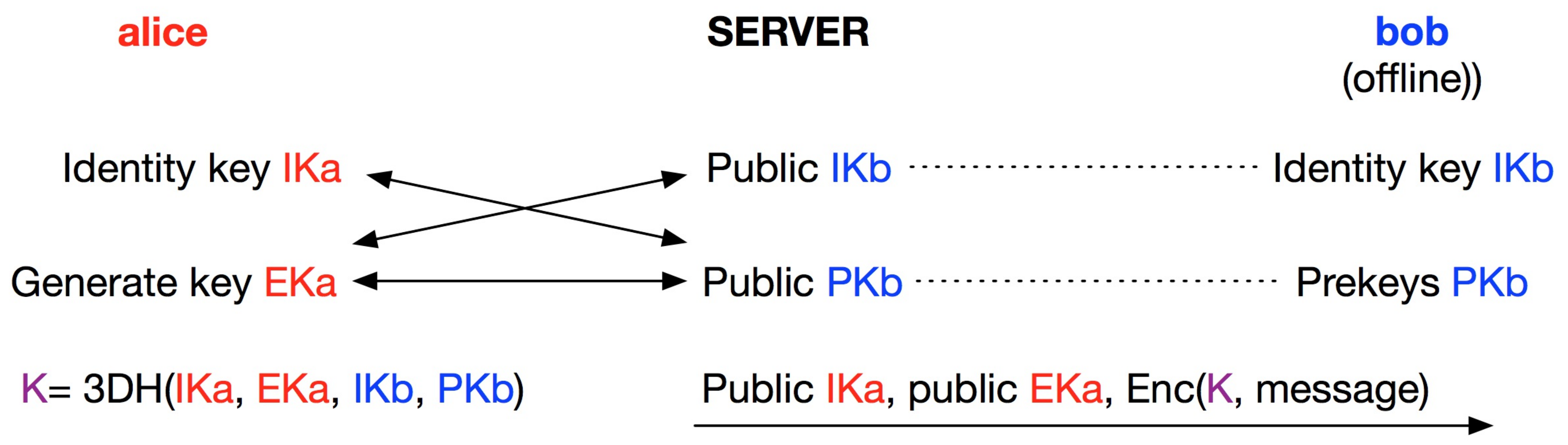


Secret Conversations relies upon the Signal Protocol. Messenger uses Signal Protocol's implementation as available in the open-source libsignal-protocol-java and libsignal-protocol-c libraries for Android and iOS respectively. Secret Conversations also incorporates new abuse-reporting features which are not present other platforms



Your personal files are encrypted!

KEY AGREEMENT: X3DH



- Combines 4 key pairs: long-term and ephemeral
- One-time **prekeys** trick, to simulate online-ness
- **Forward-secret**, resilient to malicious servers
- **Out-of-band** identity verification necessary

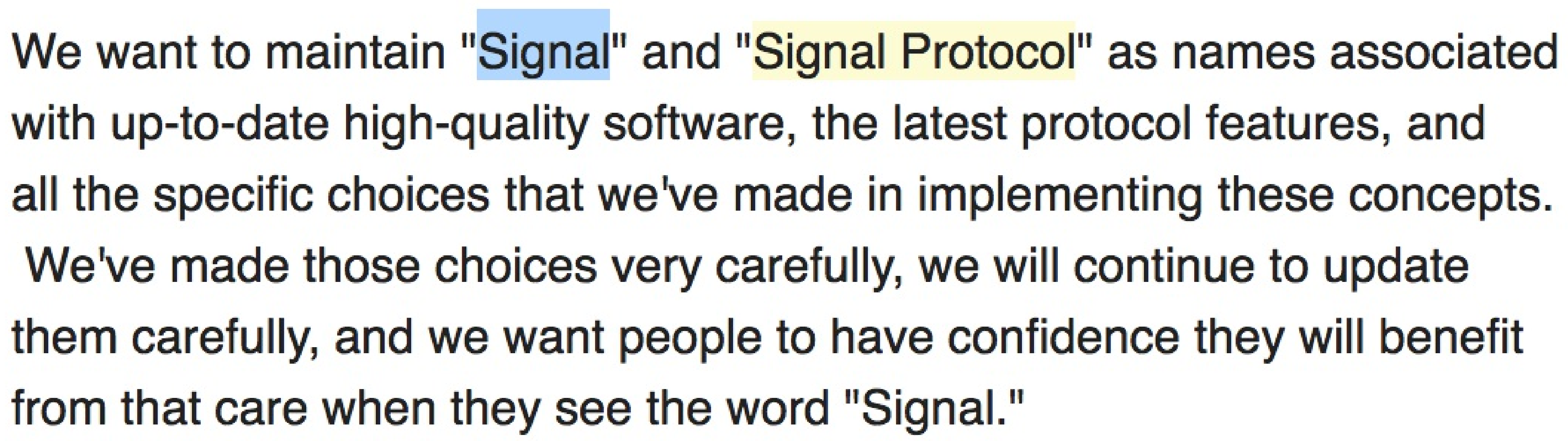
SESSION KEYS: DOUBLE RATCHET

Protocol to compute message-unique keys:

- New Diffie-Hellman for every first message from a party
- "*Key := Hash(Key)*" for consecutive messages
- **Past and future** messages safe if present key known
- **Attachments** have identical protection

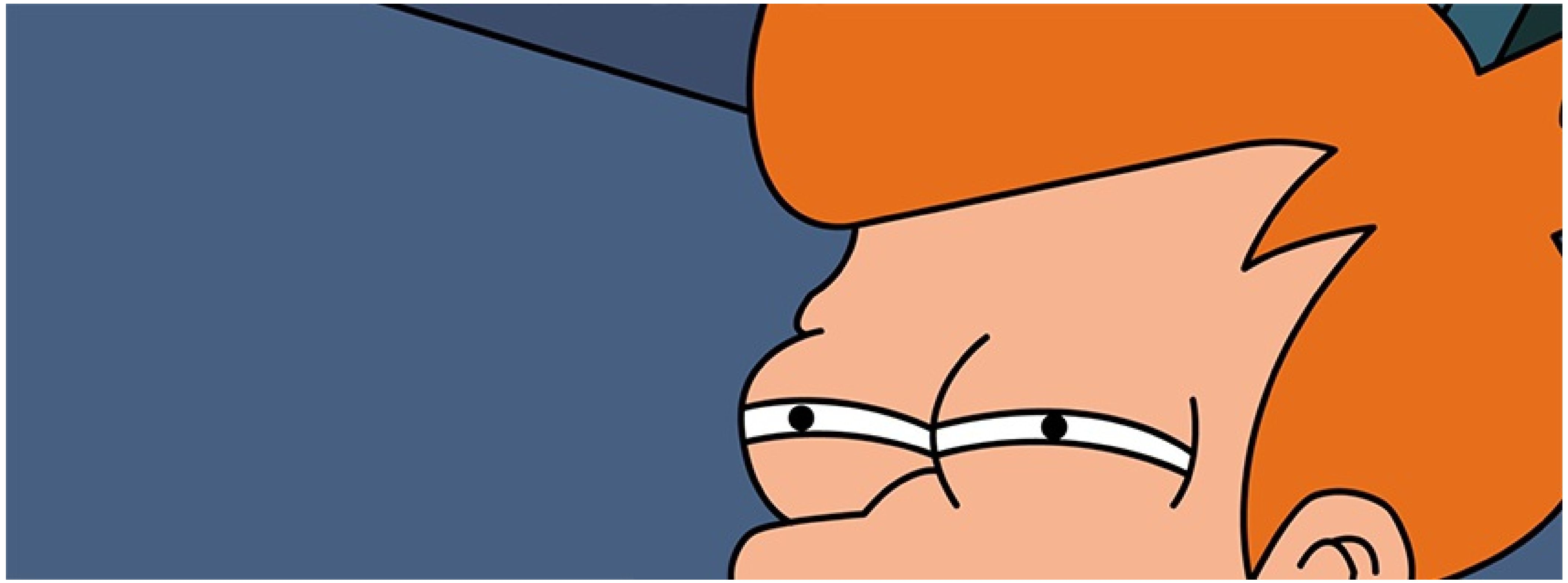
THE "SIGNAL PROTOCOL"

= X3DH and double ratchet as implemented in Signal


We want to maintain "Signal" and "Signal Protocol" as names associated with up-to-date high-quality software, the latest protocol features, and all the specific choices that we've made in implementing these concepts. We've made those choices very carefully, we will continue to update them carefully, and we want people to have confidence they will benefit from that care when they see the word "Signal."

(Moxie Marlinspike, messaging@moderncrypto.org ML, 30.11.16)

WAIT – WAS THAT ALL?



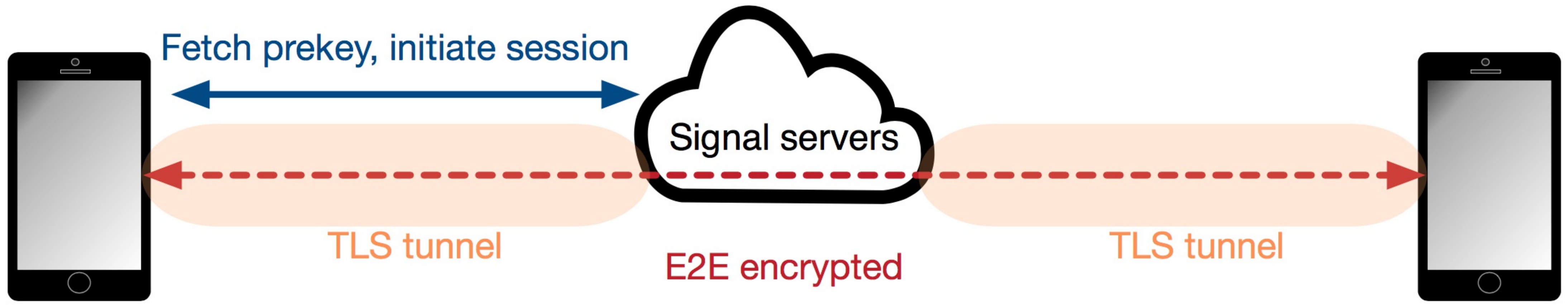
UNSPECIFIED

a.k.a. "code is documentation":

- How are attachments encrypted?
- How are audio and video streams encrypted?
- Are they fully integrity checked?
- How does group messaging work?

etc.

NETWORK ARCHITECTURE



Attachments stored on S3, at e.g.

https://whispersystems-textsecure-attachments.s3.amazonaws.com

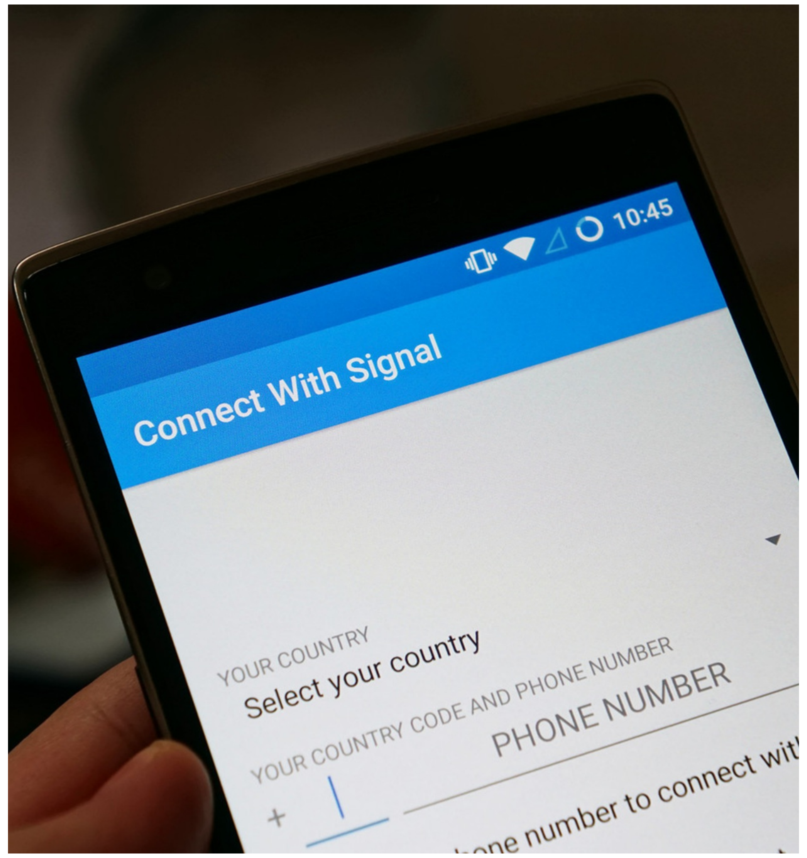
Messaging servers run by OWS

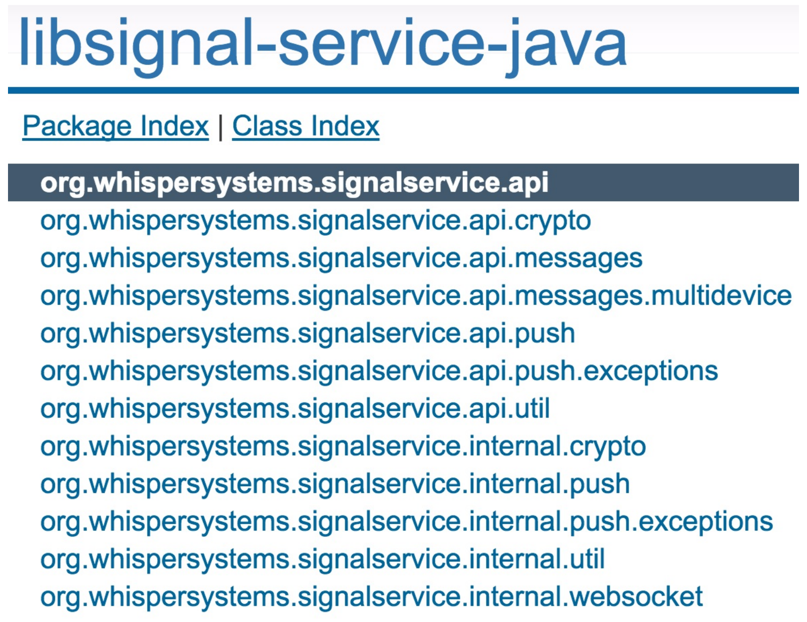
CODE BASE (CLIENT-SIDE)

Main repos from https://github.com/whisperSystems:

- **libsignal-service-java** (~20kloc Java)
- **libsignal-protocol-java** (~20kloc Java)
- **Signal-Android** (JNI + ~60kloc Java)
- **libsignal-protocol-c** (~30kloc C)
- **SignalServiceKit** (~20kloc Obj-C)
- **Signal-iOS** (~25kloc Obj-C)

ANDROID APP SOFTWARE STACK

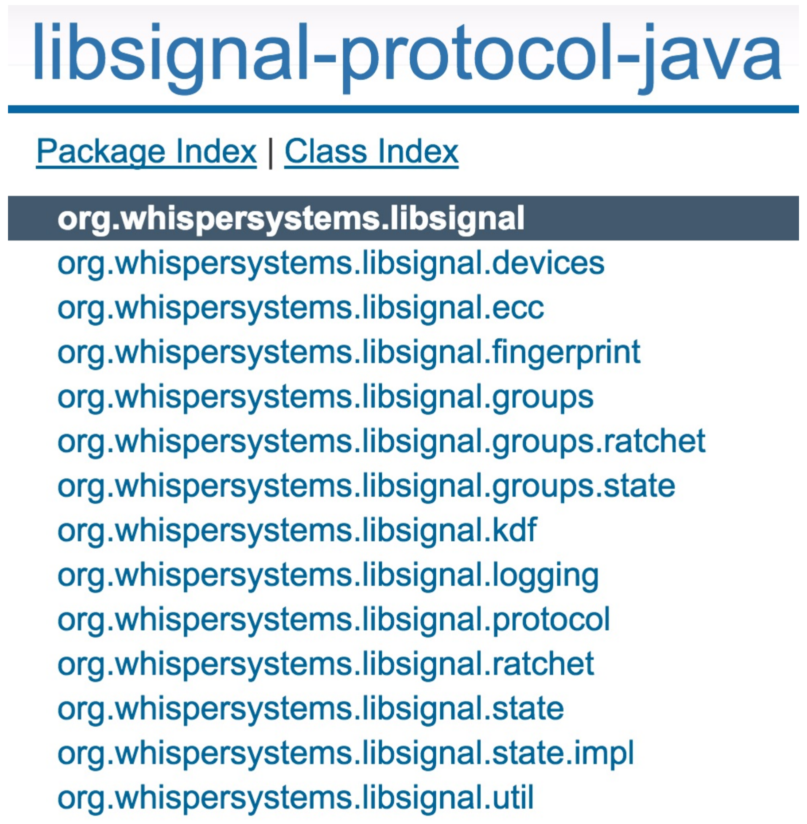




libsignal-service-java

Package Index | Class Index

org.whispersystems.signalservice.api
org.whispersystems.signalservice.api.crypto
org.whispersystems.signalservice.api.messages
org.whispersystems.signalservice.api.messages.multidevice
org.whispersystems.signalservice.api.push
org.whispersystems.signalservice.api.push.exceptions
org.whispersystems.signalservice.api.util
org.whispersystems.signalservice.internal.crypto
org.whispersystems.signalservice.internal.push
org.whispersystems.signalservice.internal.push.exceptions
org.whispersystems.signalservice.internal.util
org.whispersystems.signalservice.internal.websocket



libsignal-protocol-java

Package Index | Class Index

org.whispersystems.libsignal
org.whispersystems.libsignal.devices
org.whispersystems.libsignal.ecc
org.whispersystems.libsignal.fingerprint
org.whispersystems.libsignal.groups
org.whispersystems.libsignal.groups.ratchet
org.whispersystems.libsignal.groups.state
org.whispersystems.libsignal.kdf
org.whispersystems.libsignal.logging
org.whispersystems.libsignal.protocol
org.whispersystems.libsignal.ratchet
org.whispersystems.libsignal.state
org.whispersystems.libsignal.state.impl
org.whispersystems.libsignal.util

javax.crypto

java.security

Curve25519

PREVIOUS RESEARCH

No public record of major security bug

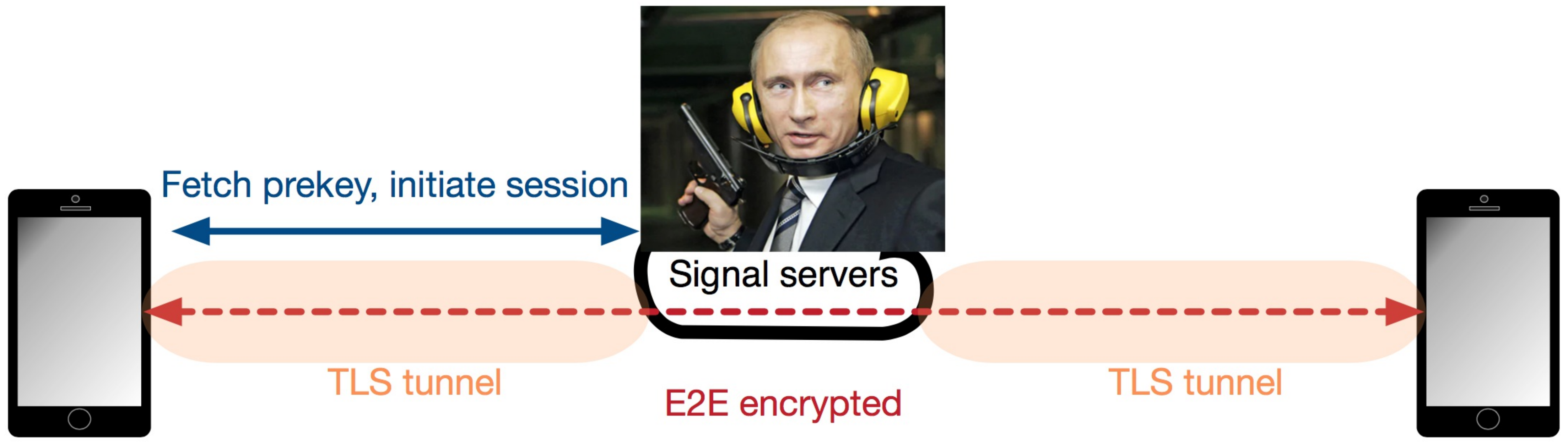
Minor security issues fixed (see tracker)

Formal analysis of the protocol (Cohn-Gordon et al.)

Key compromise impersonation, replay (Kobeissi et al.)

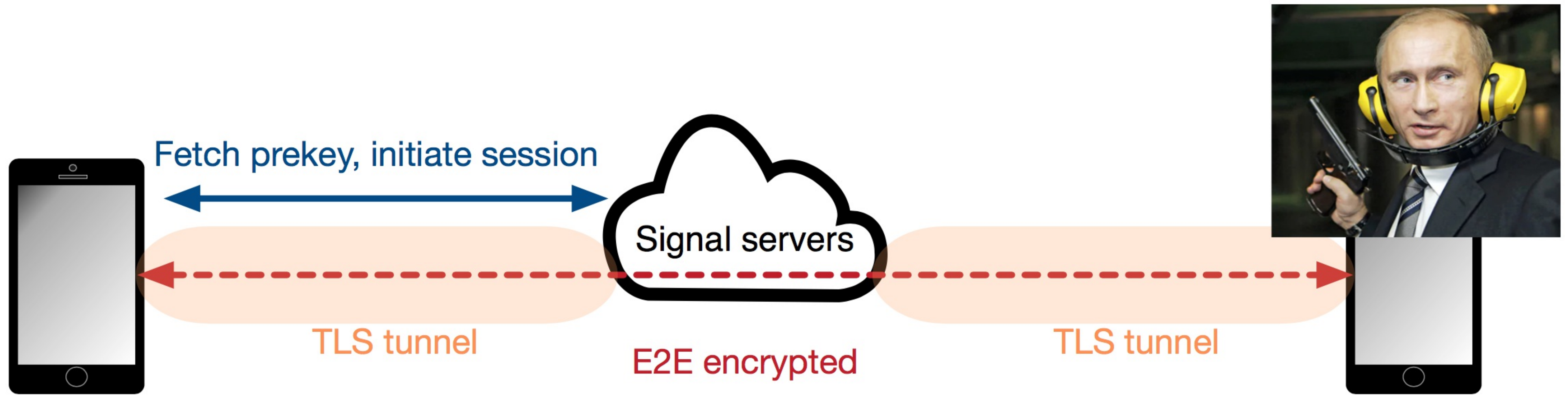
ATTACK SURFACE

THE NETWORK ATTACKER



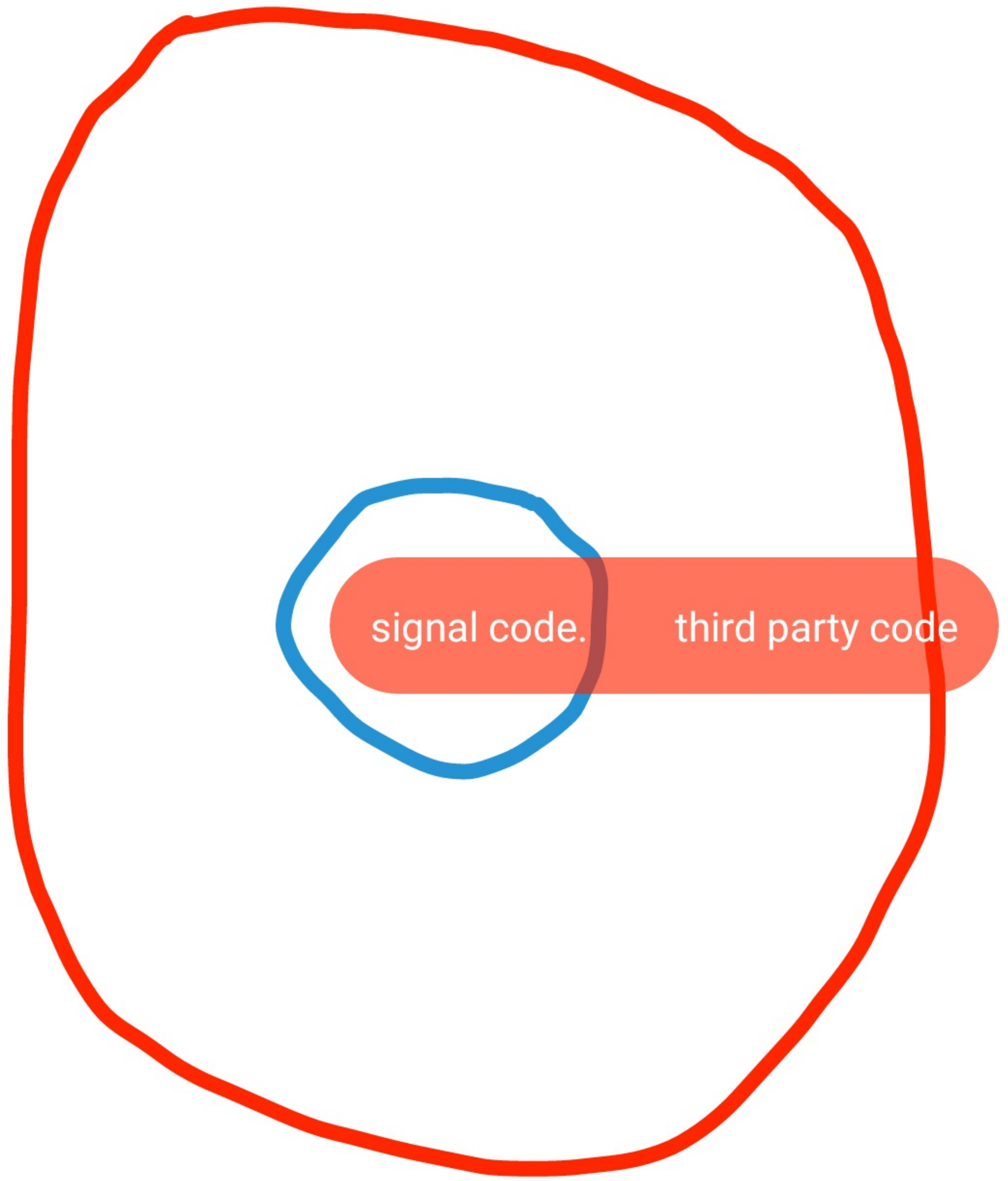
- Goal: compromise **secrecy**, **impersonate** legit peer
- Can inject/modify messages within X3DH, double ratchet
- Can sabotage **prekeys** (invalid value or format, etc.)

THE MALICIOUS PEER



- Goal: **own** other peer(s)
- **Got keys**, can trigger/abuse parsing of text/media data
- More powerful than the network attacker

DEPENDENCIES



THIRD-PARTY CODE

- **Android**: 500kloc of C etc. (WebRTC, OpenSSL)
- **iOS**: ~ 60kloc of Obj-C and C (speex codec, DSP, etc.)
- Both: OS components to decode images, low-level stuff
- Crypto: curve25519-donna.c, Java SDK crypto

MISSING MITIGATIONS AND INSECURE DEFAULTS

- No sandboxing on Android nor iOS
- Hardware keystore not used on Android
- Parsing of media files from untrusted sources
- Dependency on iOS/Android media libraries

MORE ATTACK SURFACE?

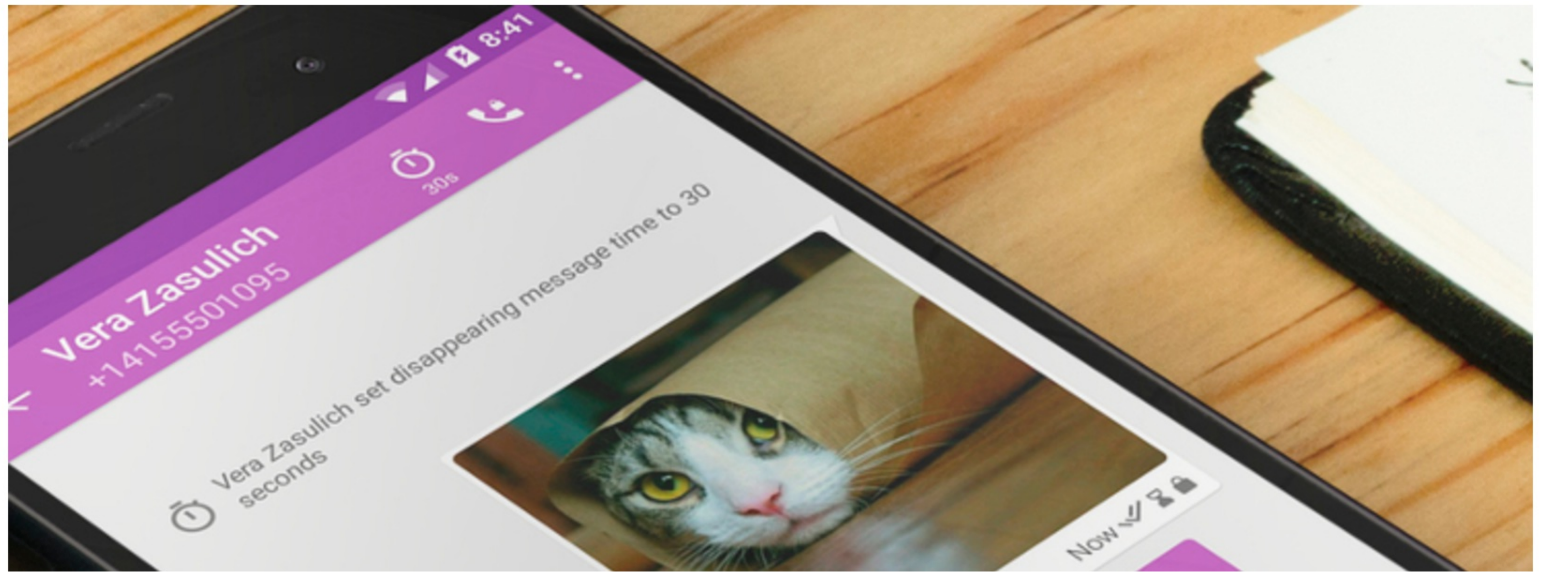
Signal and GIPHY

moxie0 on 01 Nov 2016

The latest Signal release for Android includes support for GIF search and browsing. Signal has long supported sending and receiving GIFs, but this is an experiment that allows users to browse, search, and select popular GIFs from within Signal.

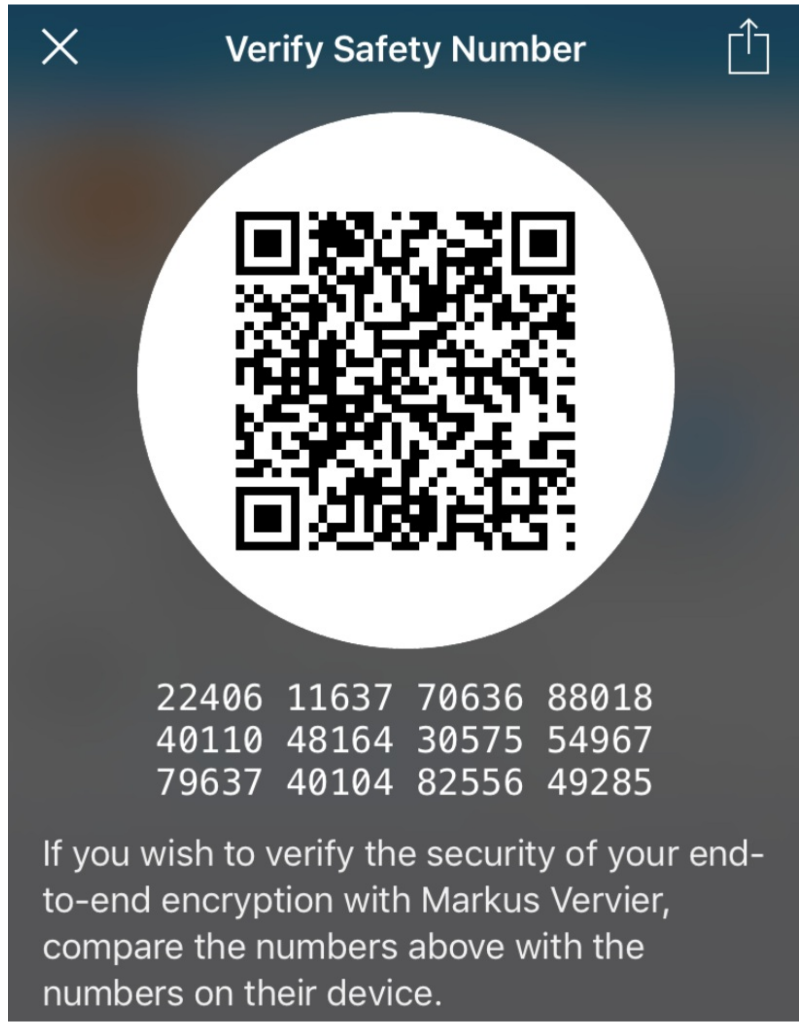
Disappearing messages for Signal

moxie0 on 11 Oct 2016



USER RESPONSIBILITIES

Check fingerprints, don't jailbreak/root, OPSEC, etc.



22406 11637 70636 88018
40110 48164 30575 54967
79637 40104 82556 49285

If you wish to verify the security of your end-to-end encryption with Markus Vervier, compare the numbers above with the numbers on their device.

UNREALISTIC SECURITY MODEL?

"Break-in recovery" protects against an attacker that **extracts temporary keys**... but only certain keys:

- Security recovered if a "KDF key" leak
- **Recovery impossible** if a "root KDF key" leaks *(Can silently MitM, as Steve Thomas tweeted)*

But keys are all in the same memory region...

Does this model make any sense on mobile?

BUGS AND "FEATURES"

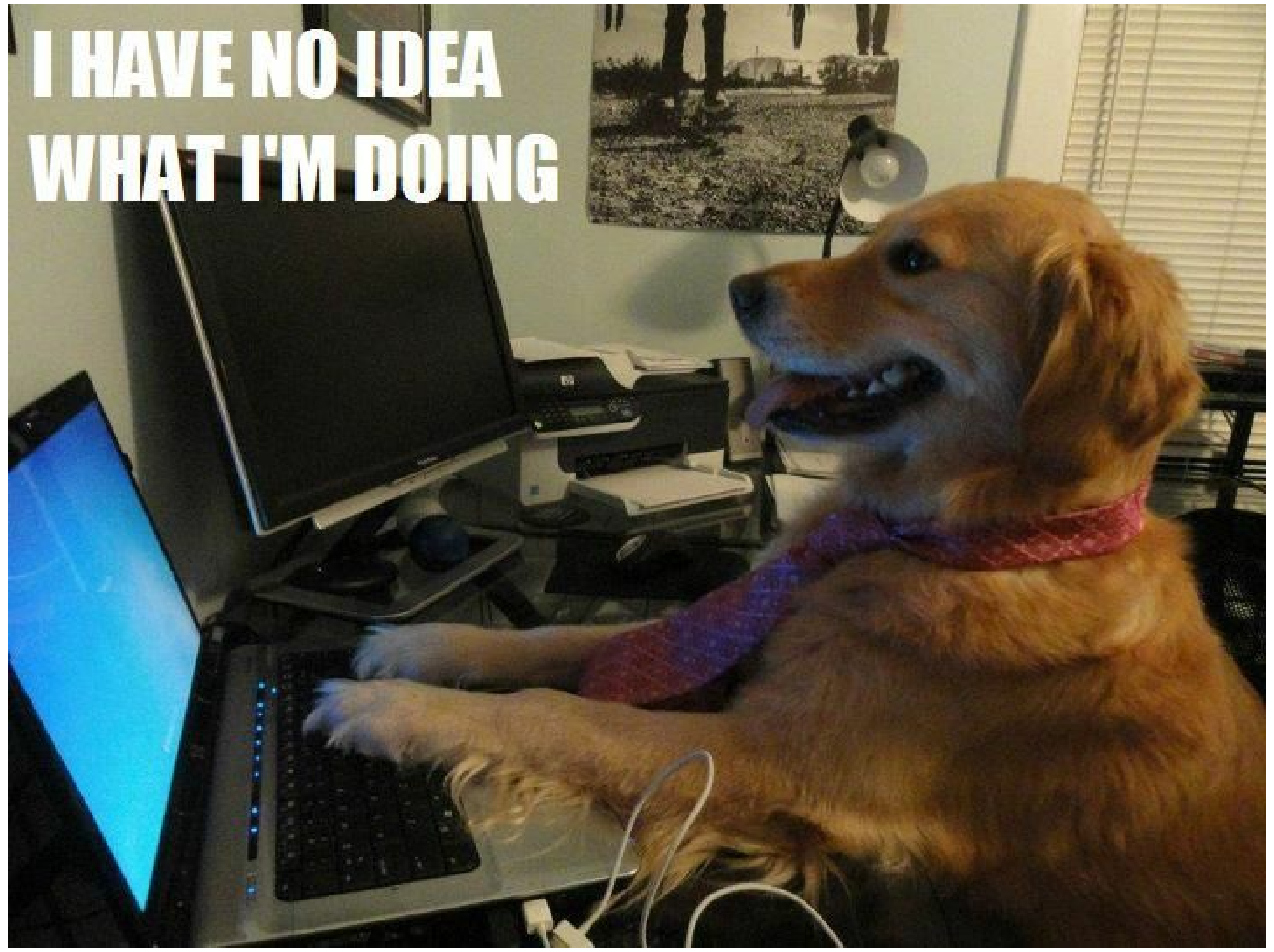
METHODOLOGY

Multi-level holistic approach to bug discovery:

- Machine learning-guided **fuzzing**
- Cloud-based parallel **concolic execution**
- State-machine **meta-model** formal verification
- **Differential cryptanalysis** using syscalls as side channels
- **Blockchain** smart contracts to record vulns found

(Releasing our tool, free for commercial use only)

ACTUAL METHODOLOGY



SERIOUSLY

No rigorous process

No automation or fuzzing

We only superficially reviewed:

- Obvious user input, protocol edge cases
- Common software bug classes
- **Client** code, *not server code*
- **Messaging** protocol/code, *not calling*

TOOLSET

iPhones, rooted Androids, Chrome extension

Signal service CLI https://github.com/AsamK/signal-cli
(to control what is sent to the server/peers)

Python MitM'ing tools

MAC BYPASS (ANDROID)

```
private void verifyMac(File file, Mac mac) throws FileNotFoundException, InvalidMacException {
  try {
    FileInputStream fin           = new FileInputStream(file);
    int             remainingData = Util.toIntExact(file.length()) - mac.getMacLength();
    byte[]          buffer        = new byte[4096];

    while (remainingData > 0) {
      int read = fin.read(buffer, 0, Math.min(buffer.length, remainingData));
      mac.update(buffer, 0, read);
      remainingData -= read;
    }
```

- 64-bit (long) file.length() cast to 32-bit (int)
- file.length() = X + 4GB => remainingData = X
- MAC computed over first X bytes => extra 4GB can be

MAC BYPASS: BASIC EXPLOITATION

MitM from S3, where attachments are stored:

- Await a request to fetch an attachment
- Pad the attachment with 4GB + use HTTP compression
- => Data attached to original data unnoticed

```
W/AttachmentDownloadJob(10484):
Caused by: javax.crypto.BadPaddingException: EVP_CipherFinal_ex
  at com.android.org.conscrypt.NativeCrypto.EVP_CipherFinal_ex(Native Method)
  at com.android.org.conscrypt.OpenSSLCipher.doFinalInternal(OpenSSLCipher.java
  at com.android.org.conscrypt.OpenSSLCipher.engineDoFinal(OpenSSLCipher.java
```

MAC BYPASS: MORE EXPLOITATION

Problem: decryption key is unknown, so can't forge meaningful ciphertext blocks..

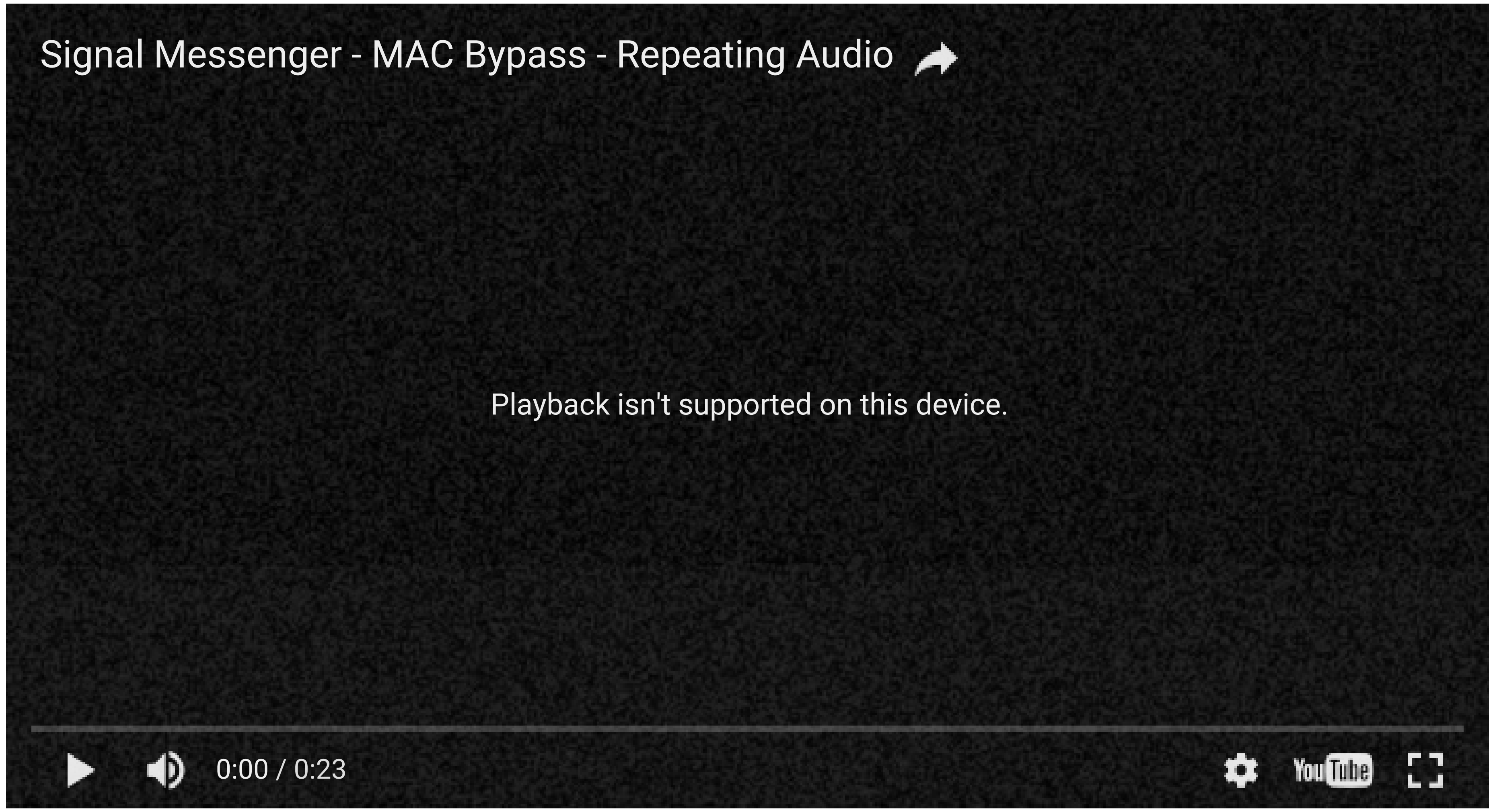
Or can we? Exploit **malleability** of CBC mode

- CBC decryption: $P[i]=Dec(C[i]) \oplus P[i-1]$
- Know/guess one $Dec(C[i])$, choose $P[i-1]$
- Control every other plaintext block!

(Image stolen from another talk)

MAC BYPASS: DEMO

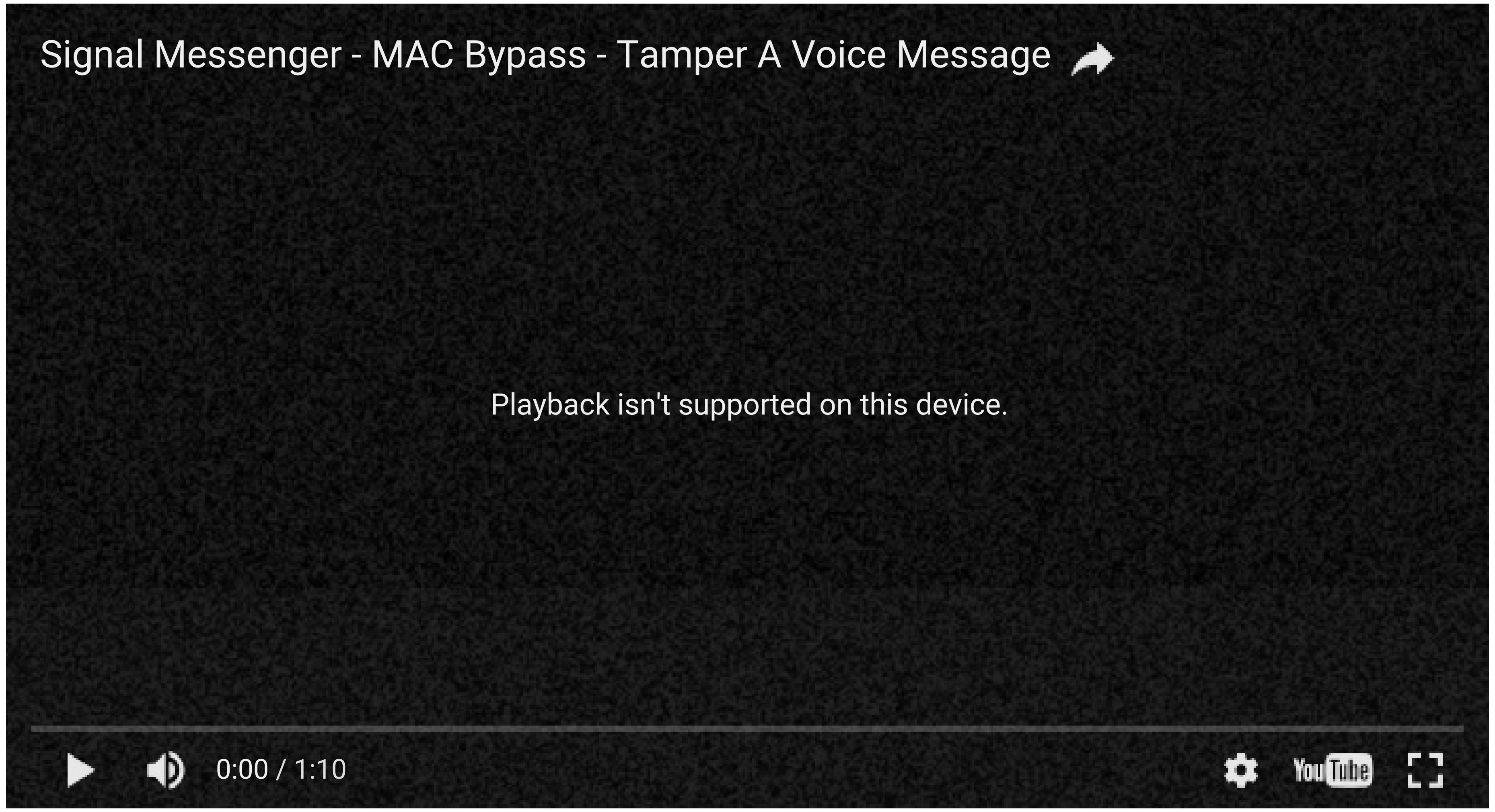
Blind message repetition


Signal Messenger - MAC Bypass - Repeating Audio

Playback isn't supported on this device.

MAC BYPASS

Known plaintext forgery


Signal Messenger - MAC Bypass - Tamper A Voice Message
Playback isn't supported on this device.
0:00 / 1:10

NO PUBLIC KEY VALIDATION

ECDH: private-key × *public-key* = shared-secret

- If *public-key* = 0, then shared-secret = 0
- Such invalid public keys should not be accepted
- Signal **accepts** *public-key* = 0

IMPACT OF INVALID KEYS

- You can force all peers to send you messages encrypted using an all-zero key (thus, **essentially in clear text**)
- **Deniability** ("PRNG bug!")
- Kills break-in recovery

C LIB CALLBACKS

C libsignal users need to define callbacks such as encrypt_func(), used to encrypt stuff (pretty important)

```
int signal_encrypt(signal_context *context,
        signal_buffer **output,
        int cipher,
        const uint8_t *key, size_t key_len,
        const uint8_t *iv, size_t iv_len,
        const uint8_t *plaintext, size_t plaintext_len)
{
    assert(context);
    assert(context->crypto_provider.encrypt_func);
    return context->crypto_provider.encrypt_func(
            output, cipher, key, key_len, iv, iv_len,
            plaintext, plaintext_len,
            context->crypto_provider.user_data);
}
```

C LIB CALLBACKS

Unit tests provide example implementations, for example to use OpenSSL to encrypt stuff in encrypt_func()

```
int test_encrypt(signal_buffer **output,
        int cipher,
        const uint8_t *key, size_t key_len,
        const uint8_t *iv, size_t iv_len,
        const uint8_t *plaintext, size_t plaintext_len,
        void *user_data)
{
    int result = 0;
    uint8_t *out_buf = 0;

    const EVP_CIPHER *evp_cipher = aes_cipher(cipher, key_len);
    if(!evp_cipher) {
        fprintf(stderr, "invalid AES mode or key size: %zu\n", key_len);
        return SG_ERR_UNKNOWN;
    }
```

BUGS IN EXAMPLE CALLBACKS

Bugs in test_encrypt():

- Type confusion => crash for certain messages (64-bit)
- Integer overflow + potential heap overflow (32-bit)

RTP PACKETS UNDERFLOW

When packetLen < sizeof(RtpHeader), payloadLen is negative => out-of-bound read in HMAC

```
RtpPacket* RtpAudioReceiver::receive(char* encodedData, int encodedDataLen) {
  int received = recv(socketFd, encodedData, encodedDataLen, 0);

  if (received == -1) {
    __android_log_print(ANDROID_LOG_WARN, TAG, "recv() failed!");
    return NULL;
  }

  RtpPacket *packet = new RtpPacket(encodedData, received);
...
RtpPacket::RtpPacket(char* packetBuf, int packetLen) {
  packet      = (char*)malloc(packetLen);
  // 1. INTEGER UNDERFLOW
  payloadLen = packetLen - sizeof(RtpHeader);
  memcpy(packet, packetBuf, packetLen);
```

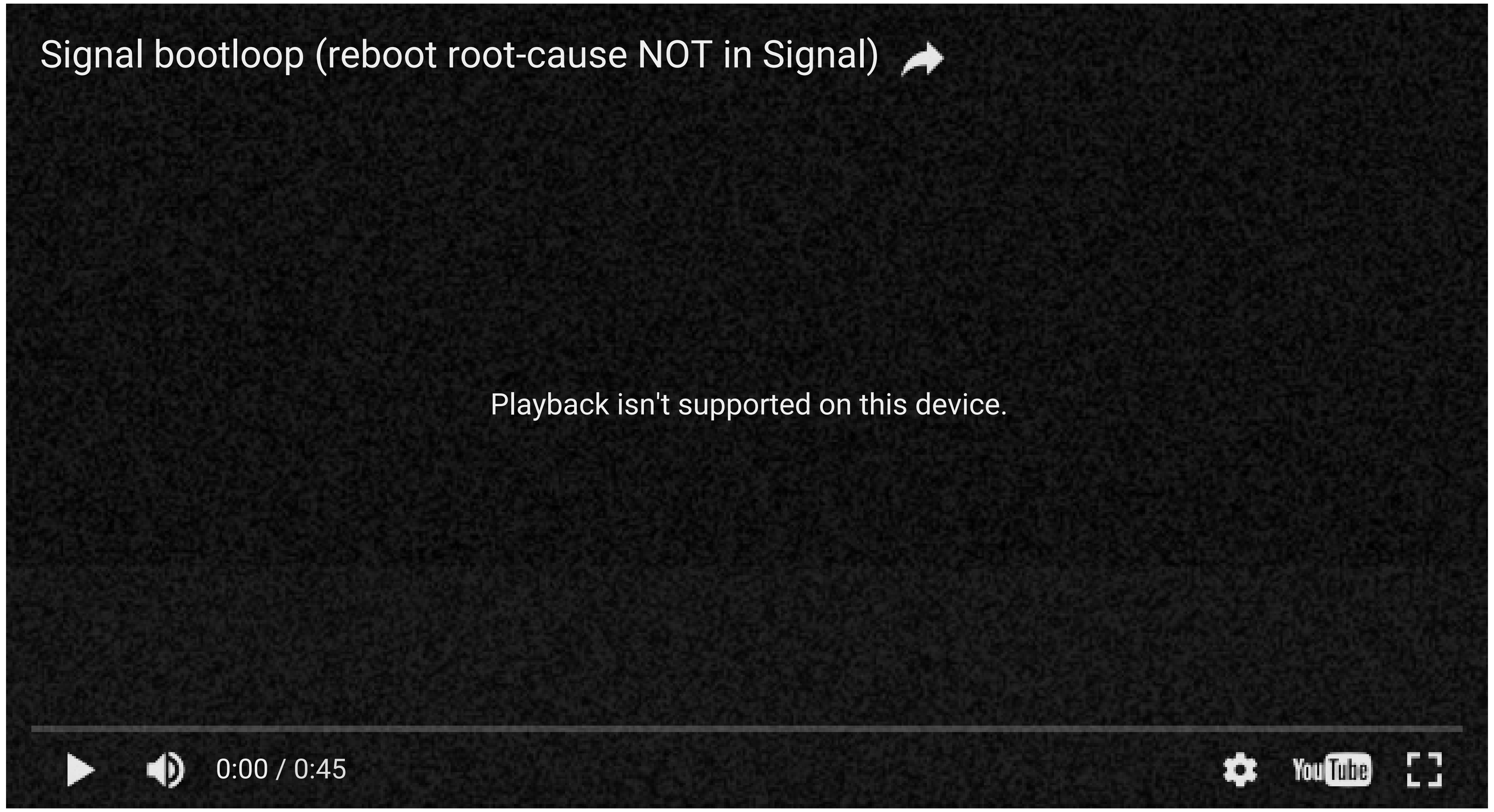
Seems unexploitable

CRASHY IMAGES

- Signal uses libskia for media decoding
- Bugs in libskia...
- Can't disable media files parsing in Signal

What can wrong?

DEMO CRASH


Signal bootloop (reboot root-cause NOT in Signal)
Playback isn't supported on this device.
0:00 / 0:45

MESSAGE REPLAY

THE EVERLASTING PREKEY

- Key agreement uses **one-time** prekeys
- Except for the **"last-resort"** key
- Fallback mechanism against DoS

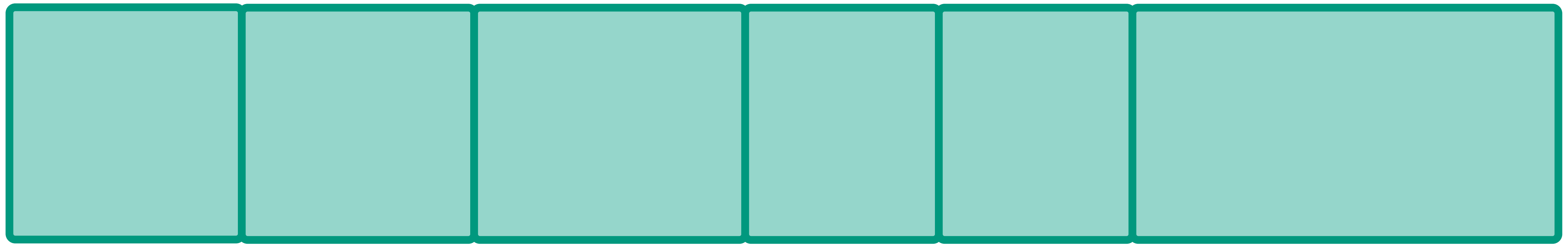
```
package org.whispersystems.libsignal.util;

public class Medium {
  public static int MAX_VALUE = 0xFFFFFF;
}
```

```
  public byte[] decrypt(PreKeySignalMessage ciphertext, DecryptionCallback callback
      throws DuplicateMessageException, LegacyMessageException, InvalidMessageE
             InvalidKeyIdException, InvalidKeyException, UntrustedIdentityException
  {
...
      if (unsignedPreKeyId.isPresent()) {
```

X3DH KEY AGREEMENT

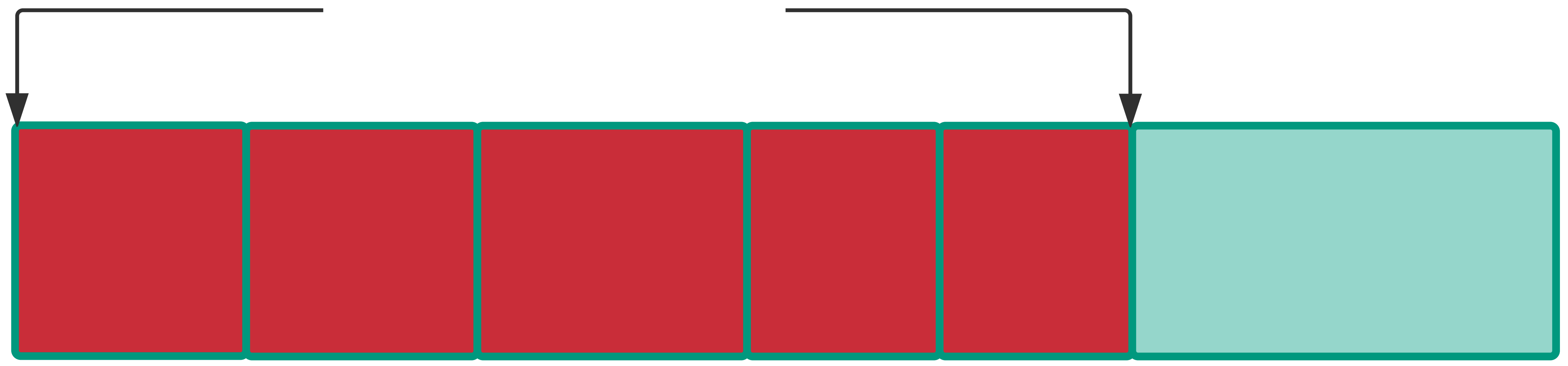
- Alice fetches Bob's id key and prekey from server...
- Computes shared secret, encrypts a message, sends with pubkeys...
- Bob computes shared secret, decrypts the message...
- Prekey removed from the server, **except if it's the last resort key** (after all prekeys have been used)

PREKEY MESSAGE STRUCTURE



Message is a bundle of a *PreKeySignalMessage* and an encrypted message (*WhisperKeyMessage*)

PREKEY MESSAGE INTEGRITY



Only the encrypted part is integrity checked!

DEFENSES AGAINST REPLAY

- Bob **won't do** new key agreement for known base keys
  - *Create fake session states and exhaust the state limit*
- A **valid ciphertext** is needed (with a valid MAC)
  - *Piggyback on messages from a different session*

WHY REPLAY IS POSSIBLE

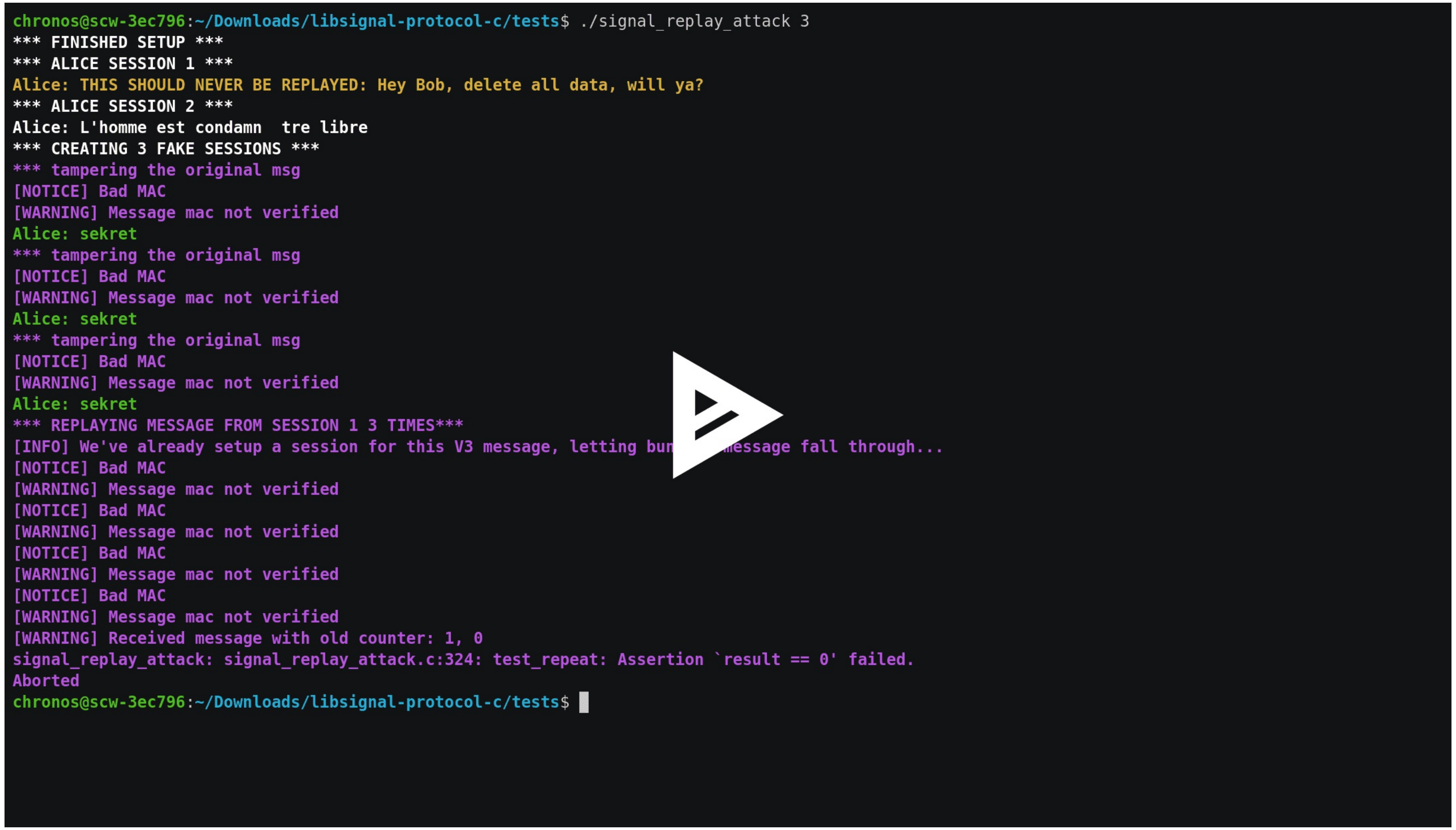
Key exchange and ciphertexts can be replayed because:

- Bob **does not check** if the encrypted message belongs to the prekey part of the message
- **Prekey messages are not integrity checked**, so a MiTM can create arbitrary session states
- **Limit of 40 session states**, old ones will be purged

HOW TO REPLAY

1. Exhaust Bob's prekeys (e.g. "evil backend" deletes normal prekeys)
2. Let Alice create a session with the last resort key
3. Record Alice's first message(s)
4. Replay! (even after Bob computes new prekeys)

REPLAY DEMO

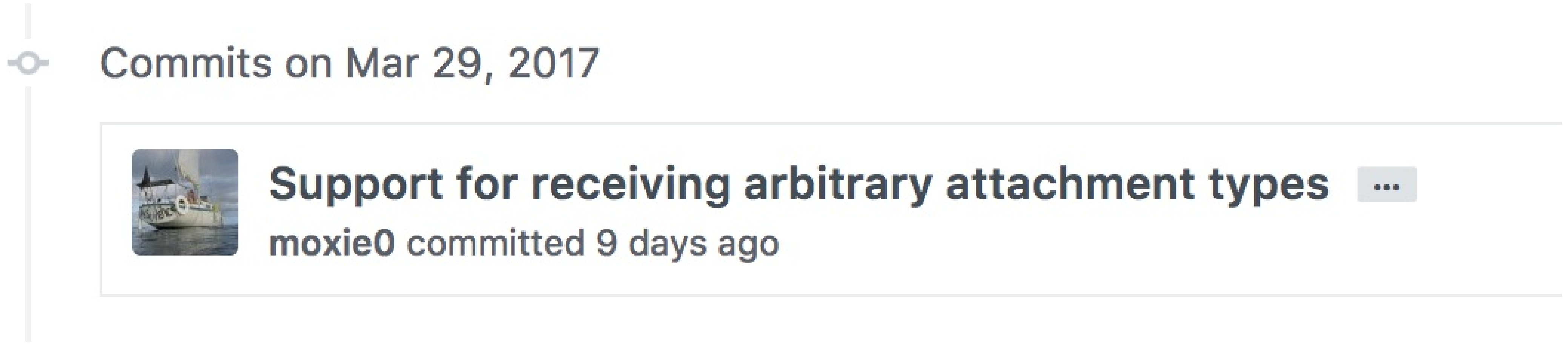


```
chronos@scw-3ec796:~/Downloads/libsignal-protocol-c/tests$ ./signal_replay_attack 3
*** FINISHED SETUP ***
*** ALICE SESSION 1 ***
Alice: THIS SHOULD NEVER BE REPLAYED: Hey Bob, delete all data, will ya?
*** ALICE SESSION 2 ***
Alice: L'homme est condamn  tre libre
*** CREATING 3 FAKE SESSIONS ***
*** tampering the original msg
[NOTICE] Bad MAC
[WARNING] Message mac not verified
Alice: sekret
*** tampering the original msg
[NOTICE] Bad MAC
[WARNING] Message mac not verified
Alice: sekret
*** tampering the original msg
[NOTICE] Bad MAC
[WARNING] Message mac not verified
Alice: sekret
*** REPLAYING MESSAGE FROM SESSION 1 3 TIMES***
[INFO] We've already setup a session for this V3 message, letting bun    message fall through...
[NOTICE] Bad MAC
[WARNING] Message mac not verified
[NOTICE] Bad MAC
[WARNING] Message mac not verified
[NOTICE] Bad MAC
[WARNING] Message mac not verified
[NOTICE] Bad MAC
[WARNING] Message mac not verified
[WARNING] Received message with old counter: 1, 0
signal_replay_attack: signal_replay_attack.c:324: test_repeat: Assertion `result == 0' failed.
Aborted
chronos@scw-3ec796:~/Downloads/libsignal-protocol-c/tests$ 
```

AUDIO FILE SERVER ON LOCALHOST

- If you play an audio file that was sent to you **an open HTTP-Server is started on localhost**
- Random 16 byte URI, random port
- Not a direct problem (unless port and URI info leaks)

MORE?



Commits on Mar 29, 2017

Support for receiving arbitrary attachment types ...

moxie0 committed 9 days ago

Looked at it yesterday morning...

- Greater risk of "friendly fire" (© Justin)
- Can coerce peers into using $K \equiv 0$
- ?

CONCLUSIONS

- Signal has a huge code base, underanalyzed
- Our work: low effort, likely missed many things
- Expecting more logic bugs, protocol edge cases, etc.
- Secure messengers need better mitigation and isolation