The Design and Implementation of Protocol-Based Hidden Key Recovery

Eu-Jin Goh, Stanford

Dan Boneh, Stanford

Philippe Golle, PARC

Benny Pinkas, HP Labs

Our contribution

- A key recovery system which is
  - Hidden
  - Unfilterable
  - Real-time
- Implemented for SSL/TLS.

Hidden Key Recovery

- Add key recovery to existing protocols *without changing protocol*.
- Modified protocol interoperates with original protocol while still leaking key.
- Protocol session is compromised if any of the involved parties supports key recovery.

Unfilterable Key Recovery

- Cannot filter key recovery channels without disrupting protocol operation.
- Even if users are aware of key recovery they cannot block it.
- Unfilterability not supported by classic covert channels.

  e.g. timing-based channels

Real Time Key Recovery

- The recovered key is the current session key.
  - Allows on-the-fly traffic decryption.
  - Well suited for wiretapping device. e.g. Carnivore

Threat - Hackers

- Break into SSL web server and patch server with hidden key recovery.
- Can eavesdrop on all secure connections to server without breaking in again.
- Original and hacked versions of the server indistinguishable from network.

Application - Governments

- Governments could pressure major software vendors to distribute SSL/SSH2 implementations with key recovery.
- Can monitor all encrypted connections where either client *or* server is compromised.

Threat – Black-box Testing

- Black-box testing
  - run the product and observe external state (network traffic)
  - Cannot detect hidden key recovery
- Must examine *source code*
  - Hard to verify large programs
  - Harder to verify hardware implementations

Model

- Client and server communicate using a standard protocol with session key K.
- Recovery agency wants to eavesdrop. Has the decryption key of a public *recovery key* K_R.
- A corrupt implementation encrypts K with recovery key K_R to generate the Escrow Agency Field (EAF).

$$EAF = E_{k_R}[K]$$

Hidden Key Recovery

- Idea - embed EAF inside protocol fields that contain random looking data.
- EAF is a ciphertext
  - indistinguishable from random to everyone except recovery agency.
- Where can the EAF be hidden?
  - random nonces, ciphertext padding

Unfilterable and Real Time Key Recovery

- **Unfilterability -** EAF is hidden in fields that are essential for correct protocol operation.

  E.g. send EAF in a field protected by MAC.
- **Real-time -** EAF delivers the key of the *current* session.

  EAF can be actual session key, or data which is sufficient for computing it.

Problem: Low Capacity Channels

- Suitable protocol fields (e.g. nonces, padding) are usually shorter than public key ciphertexts.
- EAF should be encrypted using public key encryption.
- Shortest secure pubic key scheme
  - ElGamal using elliptic curve (ECEG) over $F_{2^{163}}$.
  - Ciphertext is 41 bytes if plaintext < 20 bytes.

Low Capacity

- *What if session key > 20 bytes (max plaintext length)?*
  - Use short seed to deterministically generate session key.
  - Encrypt seed in the EAF.
  - Recovery agency can generate key from seed.

Low Capacity

- *What if available protocol fields < 41 bytes (ECEG cipher text length)?*
  1. Can embed ciphertext in several sessions.
  2. Or use ECEG to encrypt a symmetric K_s that is leaked over several sessions. Then use K_s for real-time key recovery of following sessions (using symmetric crypto).
  3. Or use weaker ECEG parameters.

SSH 2 – Padding Attack

- Padding rules => if 1 byte payload, at least 8 bytes of ciphertext from pad.
- Pad consists of random bytes => can hide 8 bytes of EAF as pad ciphertext.
- Payload, pad protected by MAC => pad ciphertext is unfilterable

SSH 2 – Key Recovery

- Typical SSH 2 network traffic pattern - large number of packets containing only single keystroke => 1 byte payload.
- Session key completely disclosed in 5 or 10 packets by either client or server.
- Attack is undetectable and unfilterable.

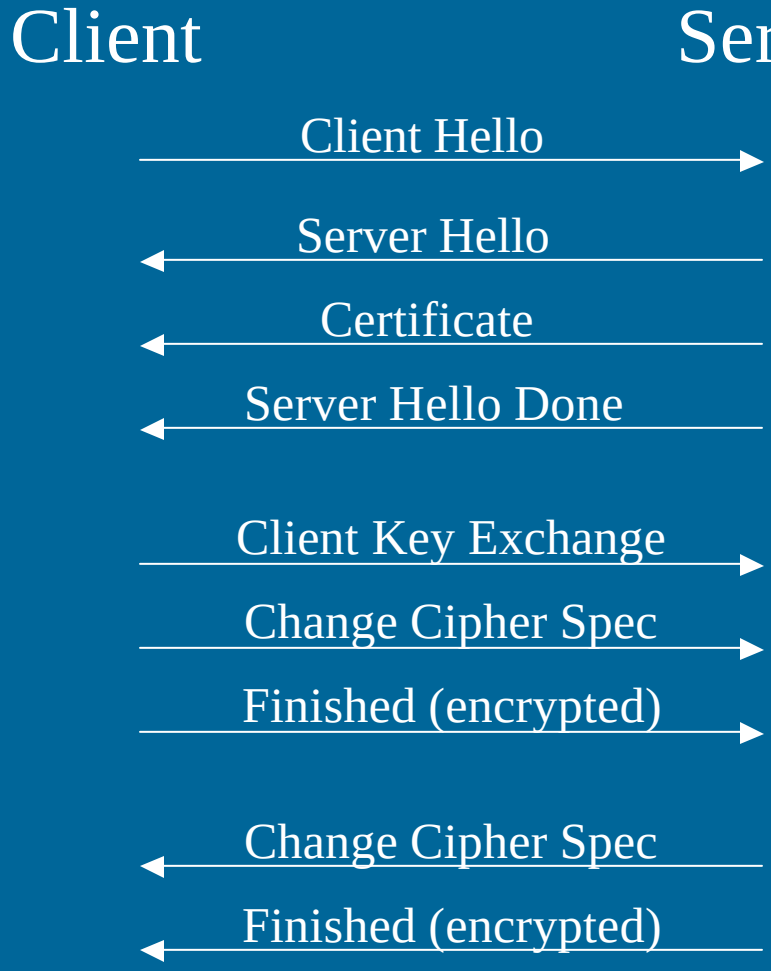
TLS Overview

- TLS – successor to SSL 3.
- TLS very similar to SSL 3 except for minor details.
- Our attack works on both TLS and SSL 3.

TLS Overview

- Two Phases in TLS:
  - Handshake Protocol negotiates cryptographic parameters.
  - Record Protocol sends application data.
- TLS - most common config is RSA key exchange and server-only auth.

TLS Handshake



Suitable Fields

Client randomness 28 bytes

Server randomness 28 bytes

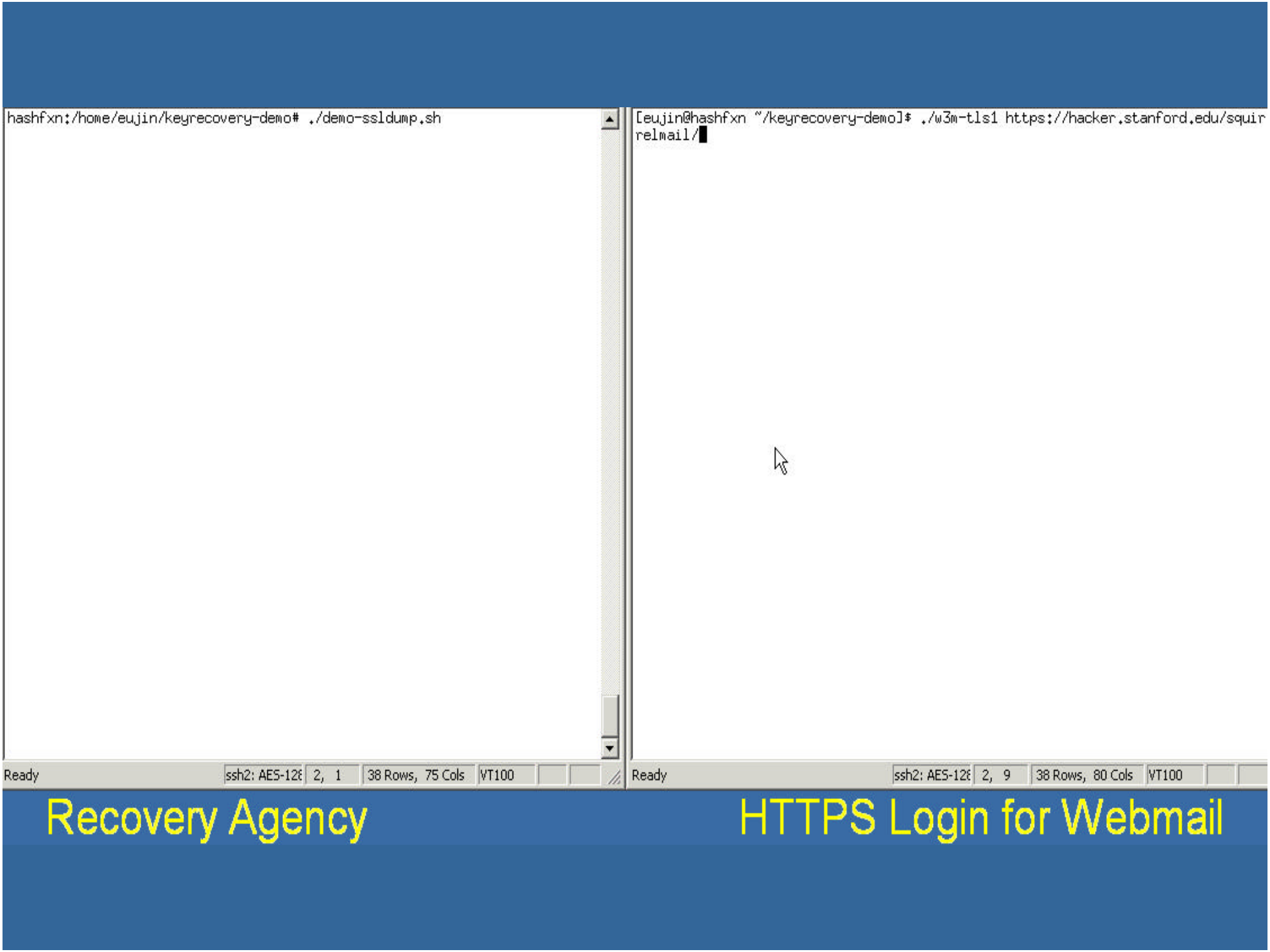
Session ID 32 bytes

Session key

A function of a client generated premaster key of length 46 bytes.

Implemented key recovery for OpenSSL

- Generate premaster secret from small seed (20 bytes) to fit in plaintext of ECEG.
- Ciphertext is 41 bytes long but the client randomness field is only 28 bytes long.
- SSL 3 - ECEG protects a symmetric key (IDEA).
  - EAF later encrypted using IDEA.
- TLS 1 - 224 bit RSA key protects EAF.


hashfxn:/home/eujin/keyrecovery-demo# ./demo-ssldump.sh
Ready
ssh2: AES-128
2, 1
38 Rows, 75 Cols
VT100
[eujin@hashfxn ~/keyrecovery-demo]$ ./w3m-tls1 https://hacker.stanford.edu/squirrelmail/
Ready
ssh2: AES-128
2, 9
38 Rows, 80 Cols
VT100
Recovery Agency
HTTPS Login for Webmail

Conclusions

- Easy to add hidden and unfilterable key recovery to existing security protocols.
  - OpenSSL – 400 lines of C for TLS and SSL
- Hard to design security protocols that resist hidden key recovery attack.