The case for ubiquitous transport-level encryption

Andrea Bittau, Michael Hamburg, Mark Handley, David Mazières, and Dan Boneh

Stanford and UCL

August 13, 2010
Goals

What would it take to encrypt the vast majority of TCP traffic?

1. **Performance.**
   - Fast enough to enable by default on almost all servers.

2. **End-point authentication.**
   - Leverage certificates, cookies, passwords, *etc.*, to achieve best possible security for any given setting.

3. **Compatibility.**
   - Works in existing networks.
   - Works with legacy apps.
Performance today can be pretty bad

Biggest problem: cost of public key cryptography.

Worst case: SSL can be 82x slower than TCP…
Performance today can be pretty bad

Biggest problem: cost of public key cryptography.

Worst case: SSL can be 82x slower than TCP…

- Worst case: tcpcrypt only 3x slower than TCP!
Problem today: app-level auth divorced from transport

1. SSL encrypts + server auth.

SSL. Authenticate server using certificates

Bank of America

If step 1 fails, step 2 doesn’t help—in fact, it harms.
Problem today:
app-level auth divorced from transport

1. SSL encrypts + server auth.

If step 1 fails, step 2 doesn’t help—in fact, it harms.
What’s the best we can do?

Level of security against a network attacker depends on scenario.

<table>
<thead>
<tr>
<th>Preconfiguration</th>
<th>Use case</th>
<th>Today’s security</th>
<th>Possible security</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>CNN</td>
<td>None</td>
<td>No passive eavesdropping</td>
</tr>
</tbody>
</table>


What’s the best we can do?

Level of security against a network attacker depends on scenario.

<table>
<thead>
<tr>
<th>Preconfiguration</th>
<th>Use case</th>
<th>Today’s security</th>
<th>Possible security</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>CNN</td>
<td>None</td>
<td>No passive eavesdropping</td>
</tr>
<tr>
<td>Server certificate</td>
<td>amazon.com.</td>
<td>Server auth</td>
<td>Server auth</td>
</tr>
</tbody>
</table>
What’s the best we can do?

Level of security against a network attacker depends on scenario.

<table>
<thead>
<tr>
<th>Preconfiguration</th>
<th>Use case</th>
<th>Today’s security</th>
<th>Possible security</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>CNN</td>
<td>None</td>
<td>No passive eavesdropping</td>
</tr>
<tr>
<td>Server certificate</td>
<td>amazon.com</td>
<td>Server auth</td>
<td>Server auth</td>
</tr>
<tr>
<td>Shared secret (cookie) no SSL</td>
<td>facebook</td>
<td>None</td>
<td>Mutual auth</td>
</tr>
</tbody>
</table>
Level of security against a network attacker depends on scenario.

<table>
<thead>
<tr>
<th>Preconfiguration</th>
<th>Use case</th>
<th>Today’s security</th>
<th>Possible security</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>CNN</td>
<td>None</td>
<td>No passive eavesdropping</td>
</tr>
<tr>
<td>Server certificate</td>
<td>amazon.com.</td>
<td>Server auth</td>
<td>Server auth</td>
</tr>
<tr>
<td>Shared secret (cookie) no SSL</td>
<td>facebook</td>
<td>None</td>
<td>Mutual auth</td>
</tr>
<tr>
<td>Shared secret and SSL</td>
<td>Bank of America</td>
<td>Mutual auth if cert and pass OK</td>
<td>Mutual auth if password OK</td>
</tr>
</tbody>
</table>
What’s the best we can do?

Level of security against a network attacker depends on scenario.

<table>
<thead>
<tr>
<th>Preconfiguration</th>
<th>Use case</th>
<th>Today’s security</th>
<th>Possible security</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>CNN</td>
<td>None</td>
<td>No passive eavesdropping</td>
</tr>
<tr>
<td>Server certificate</td>
<td>Amazon.com</td>
<td>Server auth</td>
<td>Server auth</td>
</tr>
<tr>
<td>Shared secret (cookie) no SSL</td>
<td>Facebook</td>
<td>None</td>
<td>Mutual auth</td>
</tr>
<tr>
<td>Shared secret and SSL</td>
<td>Bank of America</td>
<td>Mutual auth if cert and pass OK</td>
<td>Mutual auth if password OK</td>
</tr>
</tbody>
</table>
What’s the best we can do?

Level of security against a network attacker depends on scenario.

good with tcpcrypt

<table>
<thead>
<tr>
<th>Preconfiguration</th>
<th>Use case</th>
<th>Today’s security</th>
<th>Possible security</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>[CNN]</td>
<td>None</td>
<td>No passive eavesdropping</td>
</tr>
<tr>
<td>Server certificate</td>
<td>[amazon.com]</td>
<td>Server auth</td>
<td>Server auth</td>
</tr>
<tr>
<td>Shared secret (cookie) no SSL</td>
<td>[facebook]</td>
<td>None</td>
<td>Mutual auth</td>
</tr>
<tr>
<td>Shared secret and SSL</td>
<td>[Bank of America]</td>
<td>Mutual auth if cert and pass OK</td>
<td>Mutual auth if password OK</td>
</tr>
</tbody>
</table>
Two prevalent ways of encrypting network traffic:

1. At application layer (e.g., SSL).
   - √ Works over almost all networks.
   - × Need to modify applications.
   - × Application protocol may not allow incremental deployment.

2. At network layer (e.g., IPSec).
   - √ Works with all applications.
   - × Breaks NAT.
   - × Can’t leverage user authentication.

Ubiquitous encryption requires best of both worlds.
tcpcrypt: transport-layer encryption

tcpcrypt: a TCP option for encryption.

1. High server performance: push complexity to clients.

2. Allow applications to authenticate end points.

tcpcrypt overview

- Extend TCP in a compatible way using TCP options.
- Applications use standard BSD socket API.
- New getsockopt for authentication.
- Encryption automatically enabled if both end points support tcpcrypt.
Push expensive operations to clients

Public key operations expensive, but not all equally expensive.

RSA-exp3-2048 performance:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrypt</td>
<td>10.42</td>
</tr>
<tr>
<td>Encrypt</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Have client do decrypt

Generate ephemeral key pair

\[ \text{enc}_{\text{pubk}}(\text{master key}) \]

Without server authentication, have client decrypt.

Lets servers accept connections at 36x rate of SSL.
Session ID: hook linking tcpcrypt to app-level authentication.
- New getsockopt returns non-secret Session ID value.
- Unique for every connection (if one endpoint honest).
- If same on both ends, no man-in-the-middle.

Authenticating the Session ID authenticates the end point.
Tcpcrypt: server signs multiple session IDs at once to amortize RSA cost.

Auth example: batch signing

Tcpcrypt: server signs multiple session IDs at once to amortize RSA cost.
Tcpcrypt: server signs multiple session IDs at once to amortize RSA cost.
Auth example: batch signing

Tcpcrypt: server signs multiple session IDs at once to amortize RSA cost.

```
SID A
SID B
SID C
SID D
“A, B, C, D”
Signed by amazon.com
```

SSL servers must RSA decrypt each client's secret.

```
enc(secret A)
enc(secret B)
enc(secret C)
enc(secret D)
RSA op.
RSA op.
RSA op.
RSA op.
```
Auth example: batch signing

Tcpcrypt: server signs multiple session IDs at once to amortize RSA cost.

SSL servers must RSA decrypt each client’s secret.
Key exchange overview

Do you support tcpCrypt?

- Yes, and I support RSA

RSA public key

\[\text{enc}_{\text{pubk}}(\text{master key})\]

Generate random master key

- Clients periodically generate ephemeral public keys.
tcpcrypt key exchange

INIT1 and INIT2 too long: sent as data invisible to apps.
tcpcrypt key exchange

SYN - CRYPT(HELLO)
probe tcpcrypt

SYN ACK

ACK

tcpcrypt negotiation encoded in TCP options.
tcpcrypt key exchange

SYN - CRYPT(HELLO)
probe tcpcrypt

SYN ACK - CRYPT(PKCONF)
public key ciphers and key sizes list

ACK - CRYPT(INIT1)
symmetric ciphers and MACs list, nonce, public key

ACK - CRYPT(INIT2)
encrypted client and server nonce (master key)

tcpcrypt negotiation encoded in TCP options.

INIT1 and INIT2 too long: sent as data invisible to apps.
Key scheduling

Master key is hash of:

- Server and client nonces.
- Public key used and negotiated parameters.

Master key

- RX MAC key
- TX MAC key
- RX enc. key
- TX enc. key
- Session ID

\[ \text{hash (HMAC)} \]
Master key is hash of:

- Server and client nonces.
- Public key used and negotiated parameters.

Session caching, like in SSL: on reconnect, establish new keys without explicit key exchange.
Session caching

Low latency: completes within TCP handshake.
TCP MAC and encryption

<table>
<thead>
<tr>
<th>src port</th>
<th>dst port</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq no.</td>
<td></td>
</tr>
<tr>
<td>ack no.</td>
<td></td>
</tr>
<tr>
<td>d.off.</td>
<td>flags</td>
</tr>
<tr>
<td>window</td>
<td>checksum</td>
</tr>
<tr>
<td>options (e.g., SACK)</td>
<td>MAC option</td>
</tr>
<tr>
<td>data</td>
<td>TCP length</td>
</tr>
</tbody>
</table>

- **Allow NATs**: do not MAC ports.
- **Prevent replay**: MAC extended (implicit) seq. no.
- **Prevent truncation / extension**: MAC length.
Implementation

1. Linux kernel implementation: 4,500 LoC.

2. Portable userspace divert socket implementation: 7,000 LoC.
   - Tested on Windows (required implementing divert sockets), Mac OS, Linux and FreeBSD.

3. Binary compatible OpenSSL library that attempts tcpcrypt with batch-signing or falls back to SSL.
Performance considerations when turning encryption on:

1. Does encryption sacrifice request handling throughput? *E.g.*, how many web requests / second can a server handle?

2. Is request latency harmed? *E.g.*, How long does a client need to wait before a web page is displayed?

3. Is data throughput high? What’s the bitrate when downloading?

Hardware: 8-core, 2.66GHz Xeon (2008-era).
Software: Linux kernel implementation.
High connection rate on servers

- TCP: 98,434 connections/s
- tcpencrypt server: 27,070 connections/s
- SSL server: 754 connections/s
- No sessions cached
High connection rate on servers

<table>
<thead>
<tr>
<th></th>
<th>No sessions cached</th>
<th>All sessions cached</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>98,434</td>
<td></td>
</tr>
<tr>
<td>tcpcrypt</td>
<td>70,044</td>
<td>27,070</td>
</tr>
<tr>
<td>SSL</td>
<td>39,785</td>
<td>754</td>
</tr>
</tbody>
</table>

Connections/s

TCP tcpcrypt server SSL

98,434 70,044 39,785
Low authentication cost

- 25x faster than SSL when batch signing
Web-serve up to 25x faster than SSL

Apache serving a 44 byte static file.

- No server authentication with tcpcrypt: fair comparison would make tcpcrypt slower.
<table>
<thead>
<tr>
<th>Protocol</th>
<th>LAN connect time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>0.2</td>
</tr>
<tr>
<td>tcpcrypt cached</td>
<td>0.3</td>
</tr>
<tr>
<td>tcpcrypt not cached</td>
<td>11.3</td>
</tr>
<tr>
<td>SSL cached</td>
<td>0.7</td>
</tr>
<tr>
<td>SSL not cached</td>
<td>11.6</td>
</tr>
<tr>
<td>tcpcrypt batch sign</td>
<td>11.2</td>
</tr>
<tr>
<td>tcpcrypt CMAC</td>
<td>11.4</td>
</tr>
<tr>
<td>tcpcrypt PAKE</td>
<td>15.2</td>
</tr>
</tbody>
</table>
Lower connect latency than SSL

<table>
<thead>
<tr>
<th>Protocol</th>
<th>LAN connect time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>0.2</td>
</tr>
<tr>
<td>tcpcrypt cached</td>
<td>0.3</td>
</tr>
<tr>
<td>tcpcrypt not cached</td>
<td>11.3</td>
</tr>
<tr>
<td>SSL cached</td>
<td>0.7</td>
</tr>
<tr>
<td>SSL not cached</td>
<td>11.6</td>
</tr>
<tr>
<td>tcpcrypt batch sign</td>
<td>11.2</td>
</tr>
<tr>
<td>tcpcrypt CMAC</td>
<td>11.4</td>
</tr>
<tr>
<td>tcpcrypt PAKE</td>
<td>15.2</td>
</tr>
</tbody>
</table>
Lower connect latency than SSL

<table>
<thead>
<tr>
<th>Protocol</th>
<th>LAN connect time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>0.2</td>
</tr>
<tr>
<td>tcpcrypt cached</td>
<td>0.3</td>
</tr>
<tr>
<td>tcpcrypt not cached</td>
<td>11.3</td>
</tr>
<tr>
<td>SSL cached</td>
<td>0.7</td>
</tr>
<tr>
<td>SSL not cached</td>
<td>11.6</td>
</tr>
<tr>
<td>tcpcrypt batch sign</td>
<td>11.2</td>
</tr>
<tr>
<td>tcpcrypt CMAC</td>
<td>11.4</td>
</tr>
<tr>
<td>tcpcrypt PAKE</td>
<td>15.2</td>
</tr>
</tbody>
</table>

Batch signing does not add latency

Batch signing does not add latency

[Diagram showing network protocol handshake with timestamps and events]

**Sync signing does not add latency**

RSA decrypt start

---

Signature

**connection ready**

RSA sign start
New CPUs (Westmere) with special AES instructions can saturate 9 Gbit/s networks while encrypting.
New CPUs (Westmere) with special AES instructions can saturate 9 Gbit/s networks while encrypting.
Related work

   - Hard to integrate with application-level authentication.
   - Network compatibility issues: NATs.

2. Application layer solutions: SSL, Opportunistic encryption [Langley].
   - Poor server-side performance.
   - Requires changes to apps and possibly protocol.

3. SSL performance improvements:
   - SSL batching [Shacham & Boneh]: requires different public keys.
   - SSL rebalancing [Castelluccia, Mykletun & Tsudik]: does not leverage app-level authentication.
Conclusion

1. High server performance makes encryption a realistic default.

2. Let applications leverage tcpcrypt to maximize communication security in every setting.

3. Incrementally deployable, compatible with legacy apps, TCP and NATs.

Install tcpcrypt and help encrypt the Internet!

- [http://tcpcrypt.org](http://tcpcrypt.org)