The case for ubiquitous transport-level encryption

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

Stanford and UCL

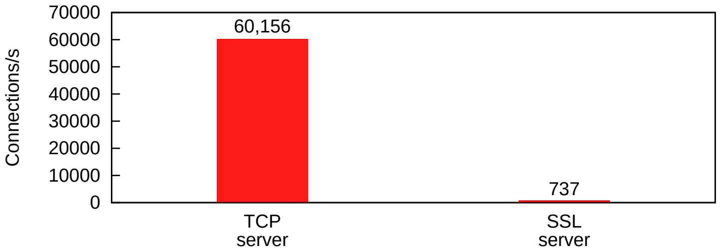
August 13, 2010

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.

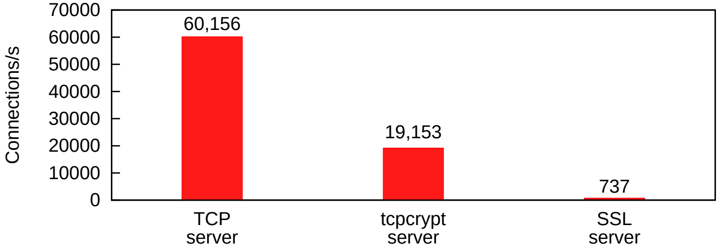
Performance today can be pretty bad



Biggest problem: cost of public key cryptography.

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

Performance today can be pretty bad



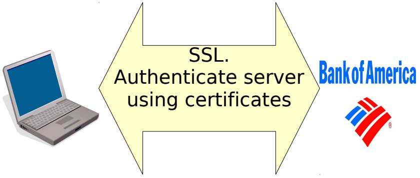
Biggest problem: cost of public key cryptography.

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

- Worst case: tcpcrypt only 3x slower than TCP!

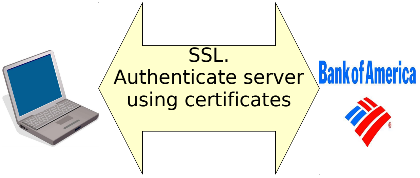
Problem today: app-level auth divorced from transport

1. SSL encrypts + server auth.

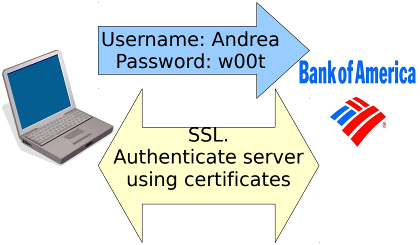


Problem today: app-level auth divorced from transport

1. SSL encrypts + server auth.



2. App auths client.



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.

Preconfiguration	Use case	Today’s security	Possible security
None	CNN	None	No passive eavesdropping

What's the best we can do?

Level of security against a network attacker depends on scenario.

Preconfiguration	Use case	Today's security	Possible security
None	CNN	None	No passive eavesdropping
Server certificate	amazon.com	Server auth	Server auth

What's the best we can do?

Level of security against a network attacker depends on scenario.

Preconfiguration	Use case	Today's security	Possible security
None	CNN	None	No passive eavesdropping
Server certificate	amazon.com	Server auth	Server auth
Shared secret (cookie) no SSL	facebook	None	Mutual auth

What's the best we can do?

Level of security against a network attacker depends on scenario.

Preconfiguration	Use case	Today's security	Possible security
None	CNN	None	No passive eavesdropping
Server certificate	amazon.com	Server auth	Server auth
Shared secret (cookie) no SSL	facebook	None	Mutual auth
Shared secret and SSL	Bank of America	Mutual auth if cert and pass OK	Mutual auth if password OK

What's the best we can do?

Level of security against a network attacker depends on scenario.

Preconfiguration	Use case	Today's security	Possible security
None	CNN	None	No passive eavesdropping
Server certificate	amazon.com	Server auth	Server auth
Shared secret (cookie) no SSL	facebook	None	Mutual auth
Shared secret and SSL	Bank of America	Mutual auth if cert and pass OK	Mutual auth if password OK

What's the best we can do?

Level of security against a network attacker depends on scenario.

goal with tcpcrypt

Preconfiguration	Use case	Today's security	Possible security
None	CNN	None	No passive eavesdropping
Server certificate	amazon.com	Server auth	Server auth
Shared secret (cookie) no SSL	facebook	None	Mutual auth
Shared secret and SSL	Bank of America	Mutual auth if cert and pass OK	Mutual auth if password OK

Backwards compatibility issues

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.
3. Backwards compatibility: all TCP apps, all networks, all authentication settings.

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.

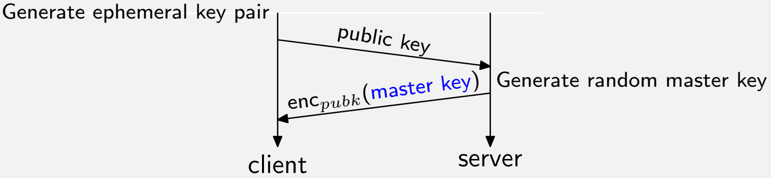
Push expensive operations to clients

Public key operations expensive, but not all equally expensive.

RSA-exp3-2048 performance:

Operation	Latency (ms)
Decrypt	10.42
Encrypt	0.26

Have client do decrypt



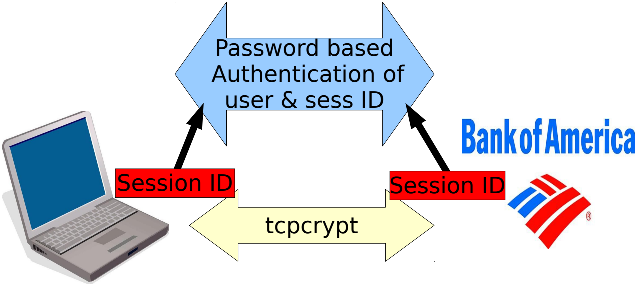
Without server authentication, have client decrypt.

Lets servers accept connections at 36x rate of SSL.

Link app auth to transport auth

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.



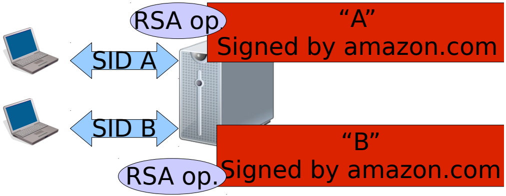
Authenticating the Session ID authenticates the end point.

Auth example: batch signing

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

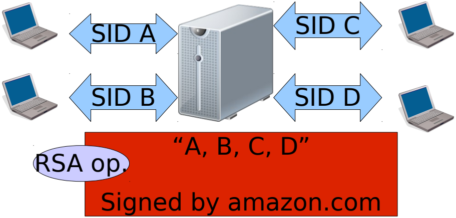
Auth example: batch signing

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



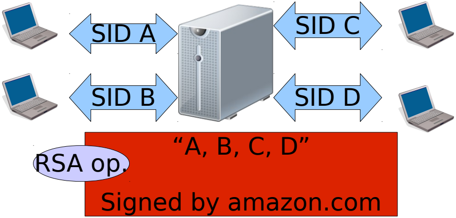
Auth example: batch signing

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

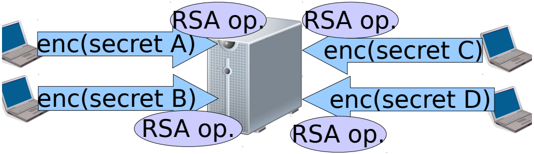


Auth example: batch signing

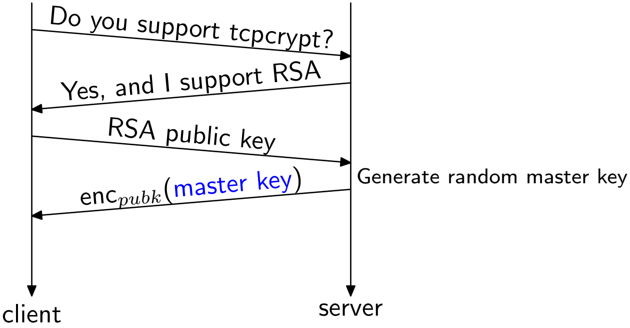
Tcpcrypt: server signs multiple session IDs at once to amortize RSA cost.



SSL servers must RSA decrypt each client's secret.

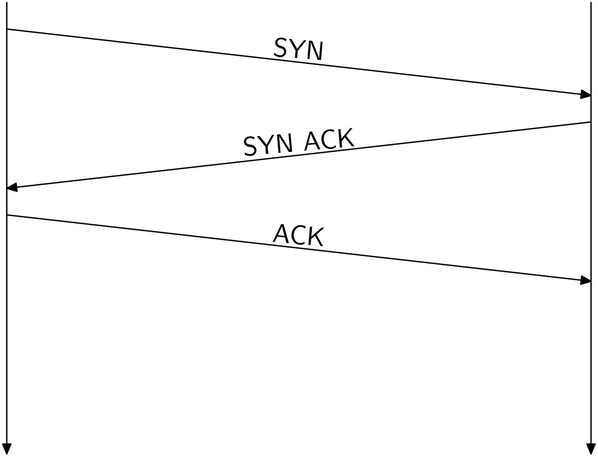


Key exchange overview

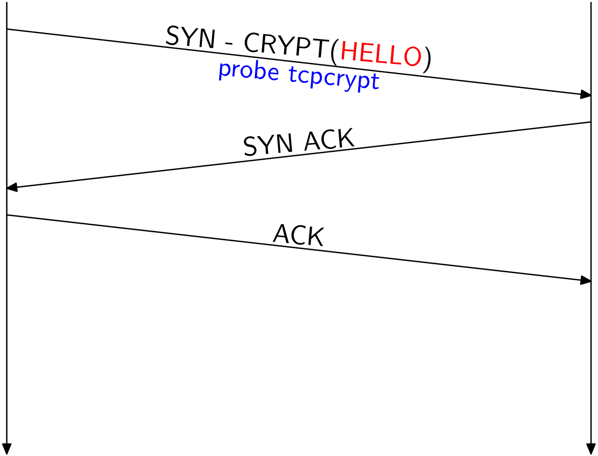


- Clients periodically generate ephemeral public keys.

tcpcrypt key exchange

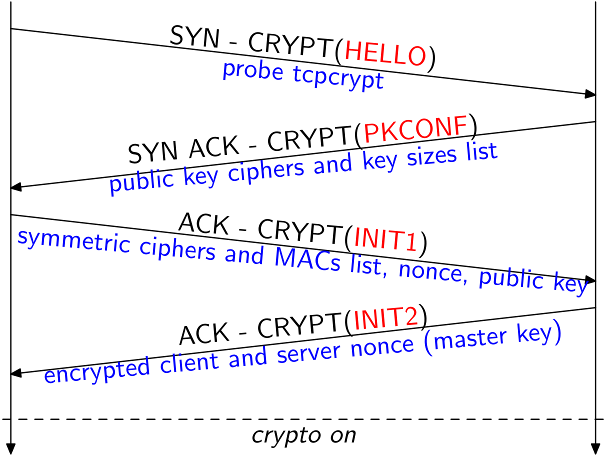


tcpcrypt key exchange



- tcpcrypt negotiation encoded in TCP options.

tcpcrypt key exchange

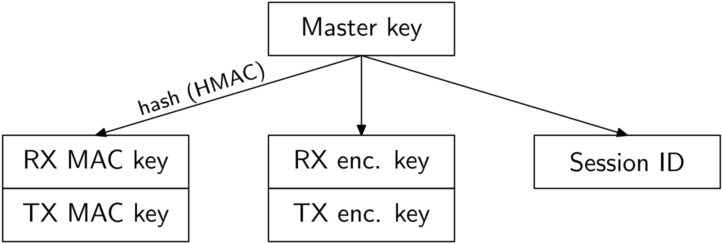


- tcpcrypt negotiation encoded in TCP options.
- INIT1 and INIT2 too long: sent as data invisible to apps.

Key scheduling

Master key is hash of:

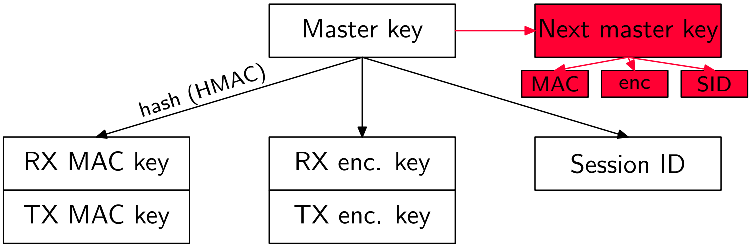
- Server and client nonces.
- Public key used and negotiated parameters.



Key scheduling

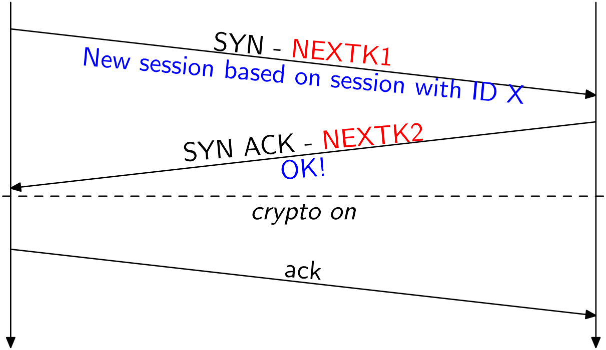
Master key is hash of:

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



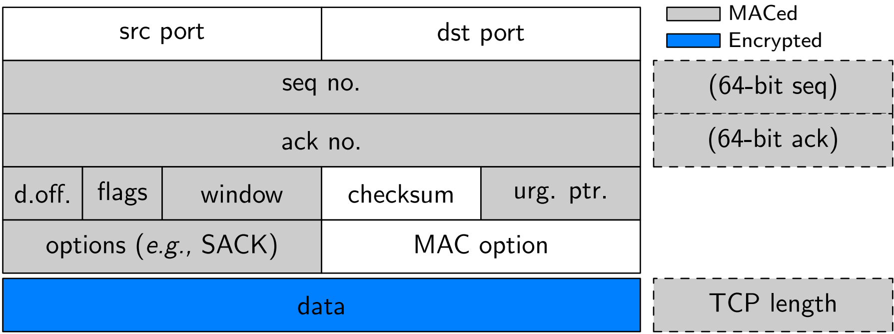
- Session caching, like in SSL: on reconnect, establish new keys without explicit key exchange.

Session caching



Low latency: completes within TCP handshake.

TCP MAC and encryption

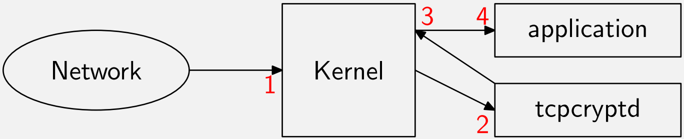


- Allow NATs: do not MAC ports.
- Prevent replay: MAC extended (implicit) seq. no.
- Prevent truncation / extension: MAC length.

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.

Packet flow in divert socket implementation.



3. Binary compatible OpenSSL library that attempts tcpcrypt with batch-signing or falls back to SSL.

Performance overview

Performance considerations when turning encryption on:

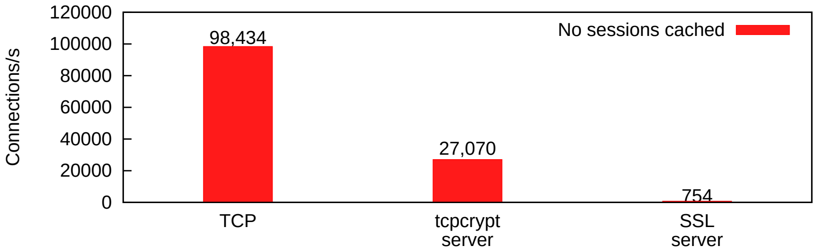
1. Does encryption sacrifice request handling throughput? *E.g.*, how many web requests / second can a server handle?

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

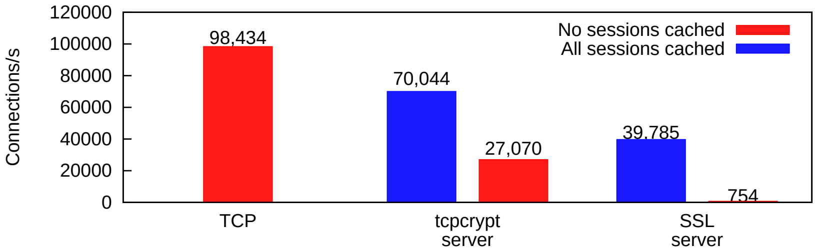
3. Is data throughput high? What's the bitrate when downloading?

Hardware: 8-core, 2.66GHz Xeon (2008-era).
Software: Linux kernel implementation.

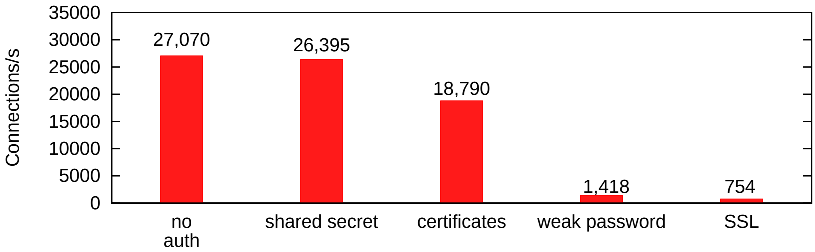
High connection rate on servers



High connection rate on servers

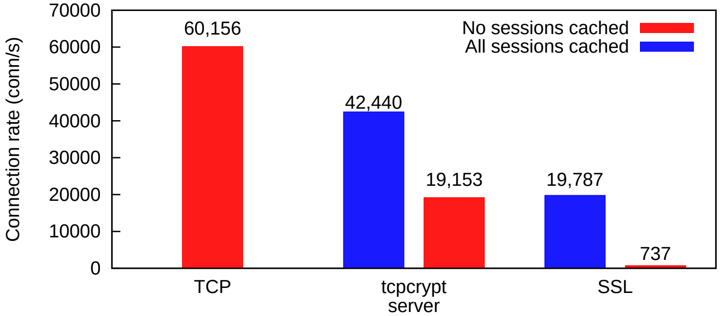


Low authentication cost



- 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.

Lower connect latency than SSL

Protocol	LAN connect time (ms)
TCP	0.2
tcpcrypt cached	0.3
tcpcrypt not cached	11.3
SSL cached	0.7
SSL not cached	11.6
tcpcrypt batch sign	11.2
tcpcrypt CMAC	11.4
tcpcrypt PAKE	15.2

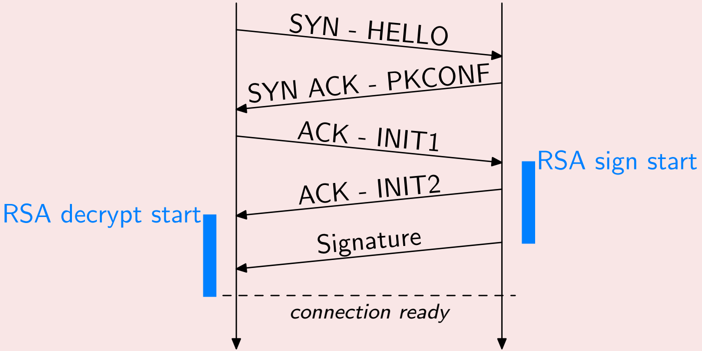
Lower connect latency than SSL

Protocol	LAN connect time (ms)
TCP	0.2
tcpcrypt cached	0.3
tcpcrypt not cached	11.3
SSL cached	0.7
SSL not cached	11.6
tcpcrypt batch sign	11.2
tcpcrypt CMAC	11.4
tcpcrypt PAKE	15.2

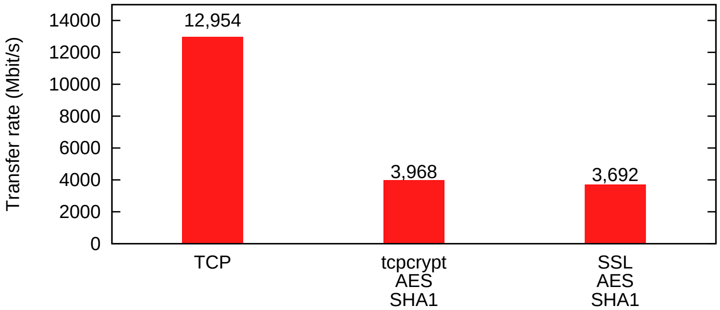
Lower connect latency than SSL

Protocol	LAN connect time (ms)
TCP	0.2

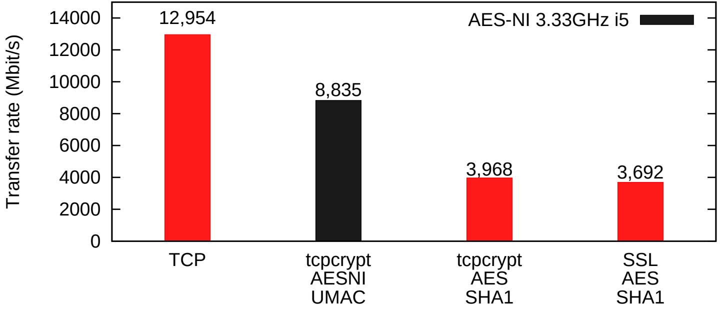
Batch signing does not add latency



Gigabit encryption possible



Gigabit encryption possible



New CPUs (Westmere) with special AES instructions can saturate 9 Gbit/s networks while encrypting.

Related work

1. Network layer solutions: IPSec, Better Than Nothing Security.
   - 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`