tcpcrypt: real transport-level encryption

Andrea Bittau, Mike Hamburg, Mark Handley, David Mazieres, Dan Boneh.

UCL and Stanford.

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

Performance

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

Authentication

- Leverage certificates, cookies, passwords, etc., to give best possible security for any given setting.

Compatibility

- Works in existing networks
- Works with unmodified legacy applications

An observation on layering of crypto

- Encryption is a generic function.
  - Independent of the semantics of the application.
- Integrity Protection is a generic function.
  - What arrives should be what is sent.
- Authentication is strongly application-specific.
  - Depends on the semantics of the application.

An observation on layering of crypto

<u>Observation:</u> **Encryption** and **Integrity Protection** are lower-layer functions than **Authentication**.

- **Encryption** and Integrity Protection are natural <u>transport-layer</u> functions.
  - Cannot integrity-protect transport protocol from above it.
  - Different transport sessions have different security requirements so cannot share encryption keys.
- **Authentication** is <u>application-layer</u>.

Tcpcrypt

Tcpcrypt uses TCP options to provide deployable transport-level encryption.

- High server performance - push complexity to clients
- Allow applications to authenticate endpoints.
- Backwards compatibility: all TCP apps, all networks, all authentication settings.

Tcpcrypt overview

- Extend TCP in a compatible way using TCP options.
- Existing applications use standard socket API, just like regular TCP.
  - Encryption automatically enabled if both end points support Tcpcrypt.
- Extended applications can use a new **getsockopt()** for authentication.

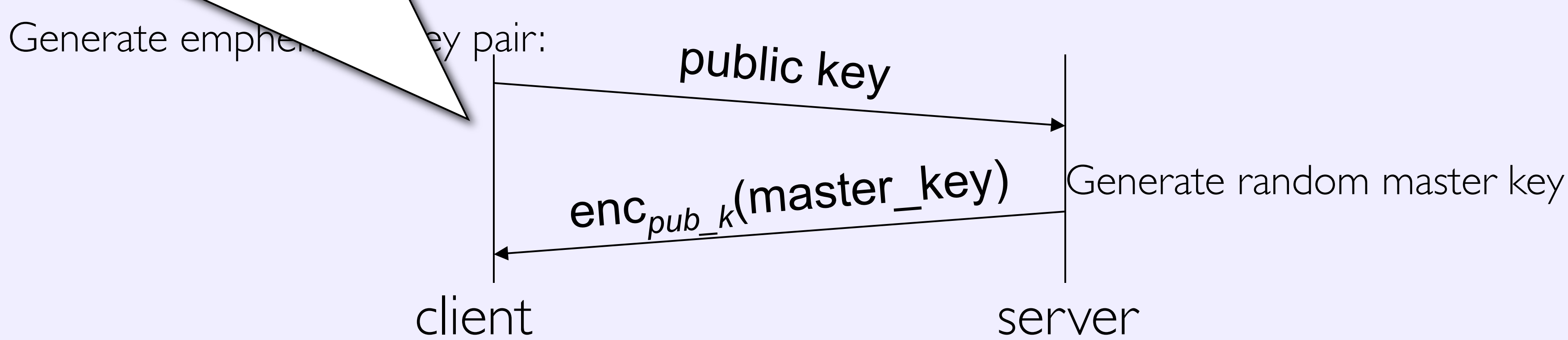
Push expensive operations to the client

Initial handshake:
No authentication.
Client decrypts.

Lets servers accept connections
36x faster than SSL

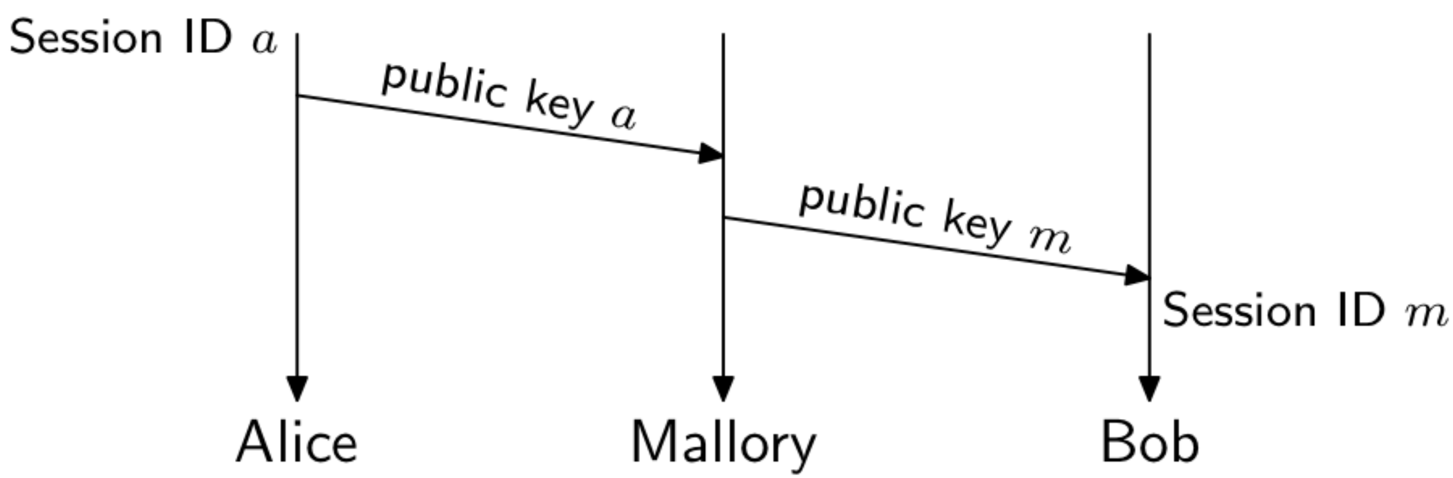
eration	Latency
rypt	0.26ms
crypt	10.42ms

Po

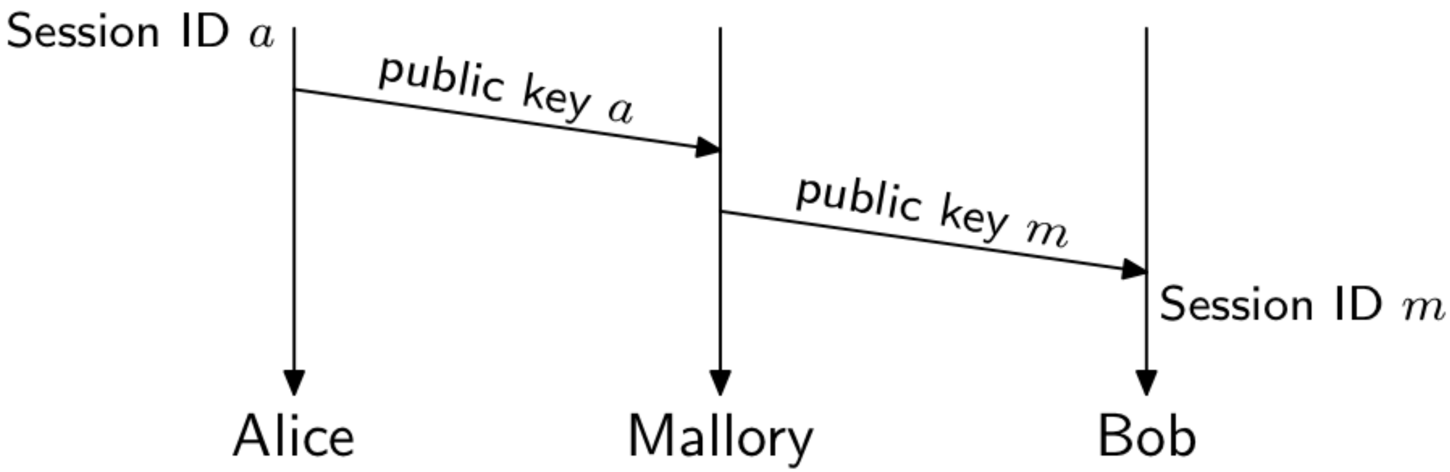


After initial handshake, tcpcrypt's Session ID provides the hook to link application authentication to the session.

- New getsockopt() returns non-secret Session ID value.
- Unique for every connection.
- If same on both ends, guaranteed there's no man-in-the-middle.



How to check the Session ID?

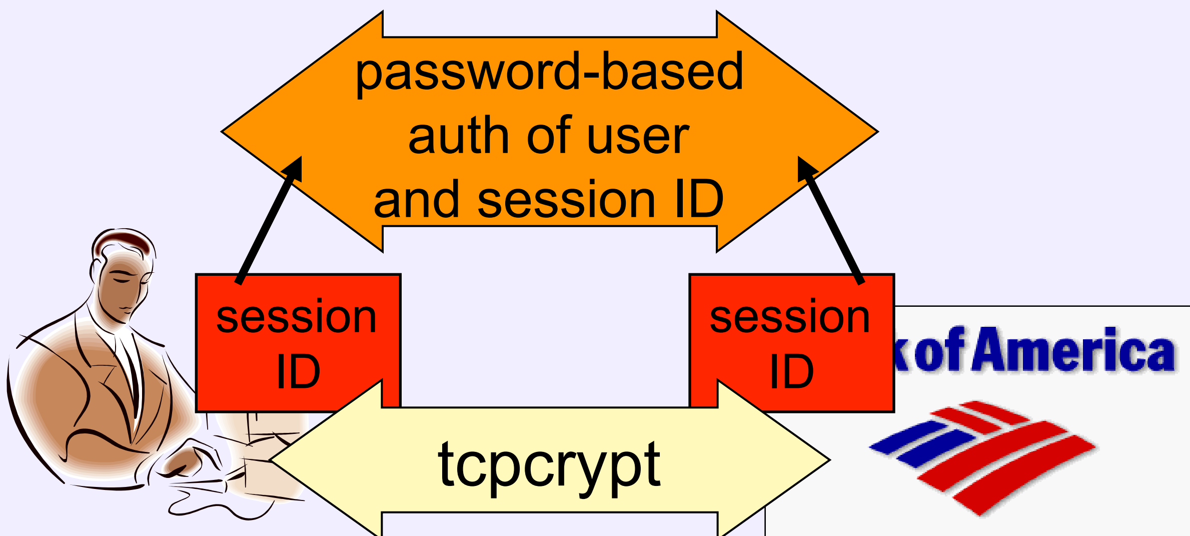


- **Out-of-band:** e.g., phone call, other secure protocol.
- **PKI:** server signs Session ID.
- **Pre-shared secret:** send CMAC of Session ID, keyed with Pre-shared secret.

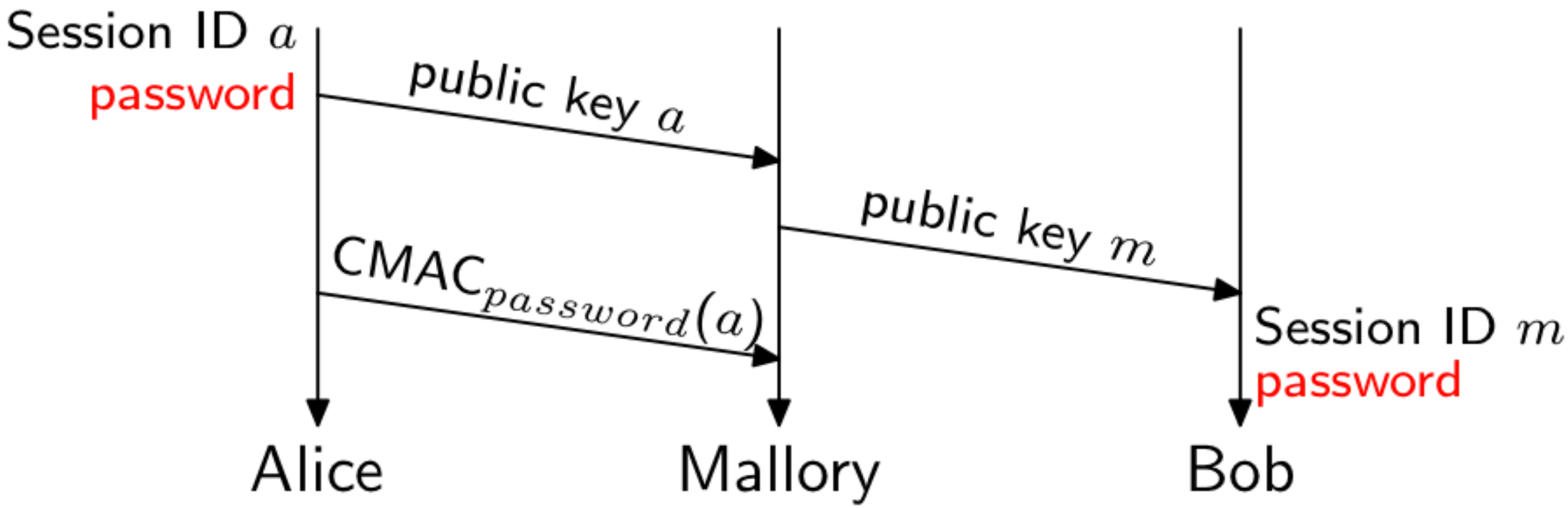
Authentication Example: Password-based Mutual Authentication

- Whenever a user knows a password, mutual authentication should be used.
  - Does not rely on user to spot spurious URLs.

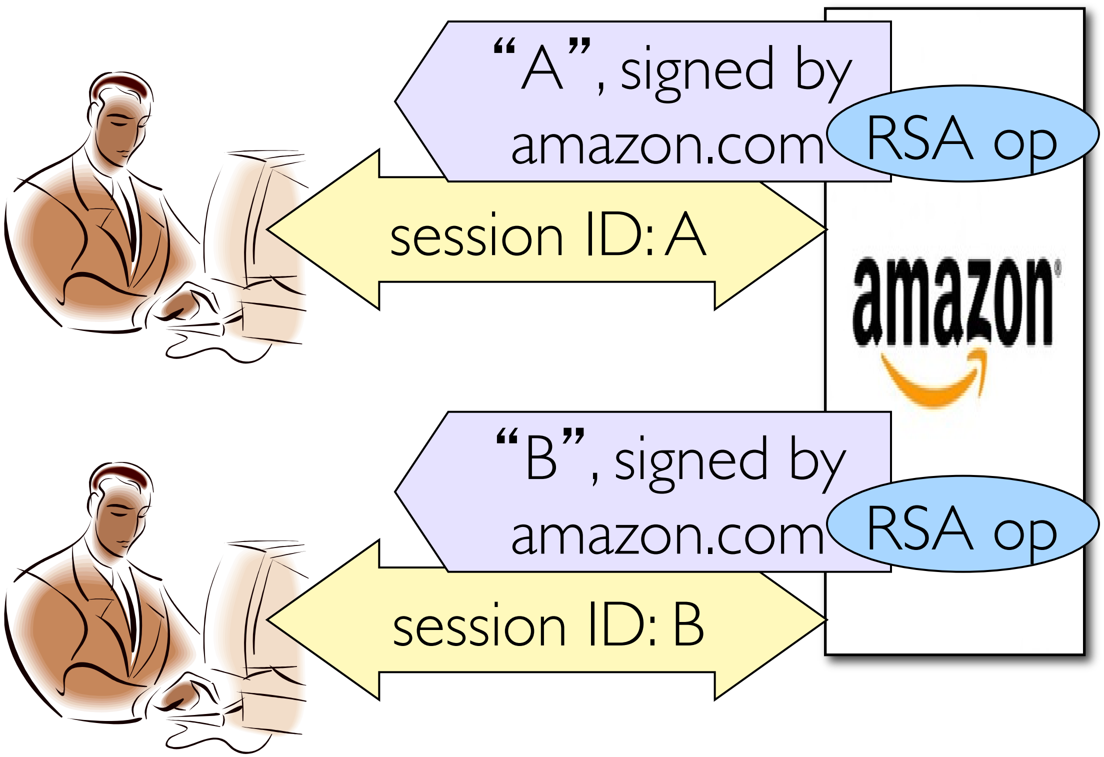
Authenticating the session ID authenticates the endpoint



Authentication Example: Password-based Mutual Authentication

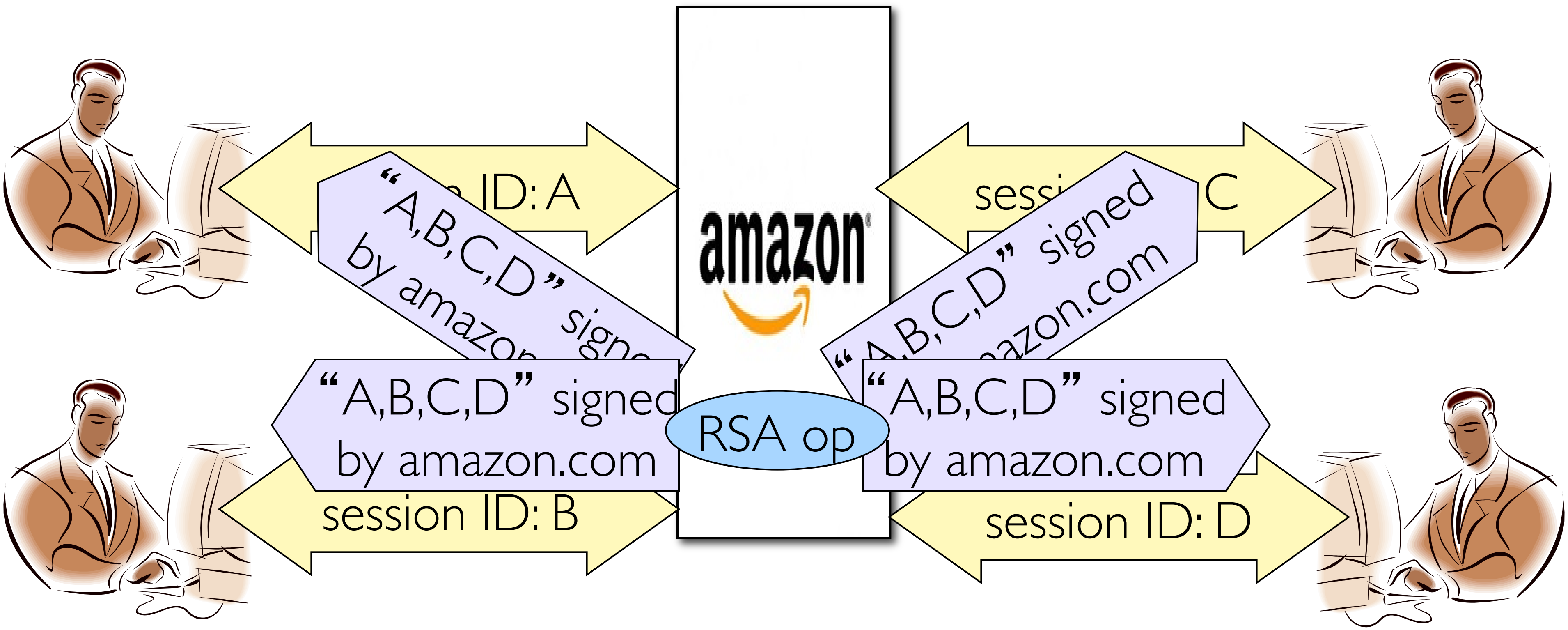


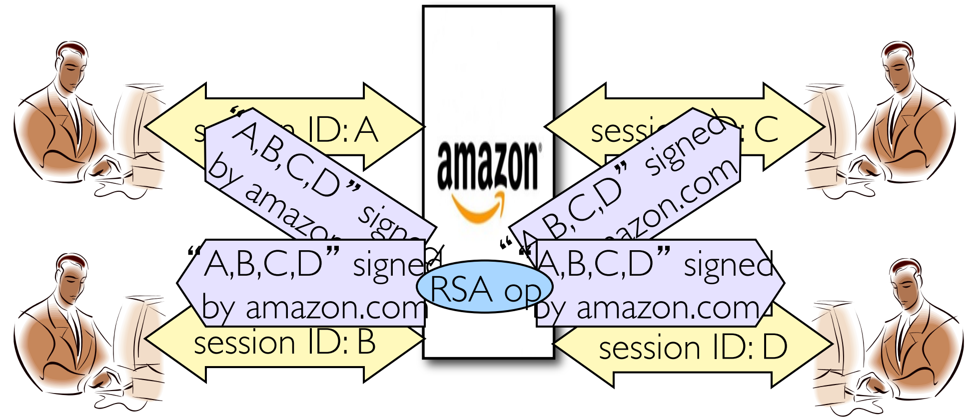
Authentication Example:
Signing a batch of session IDs to amortize RSA costs



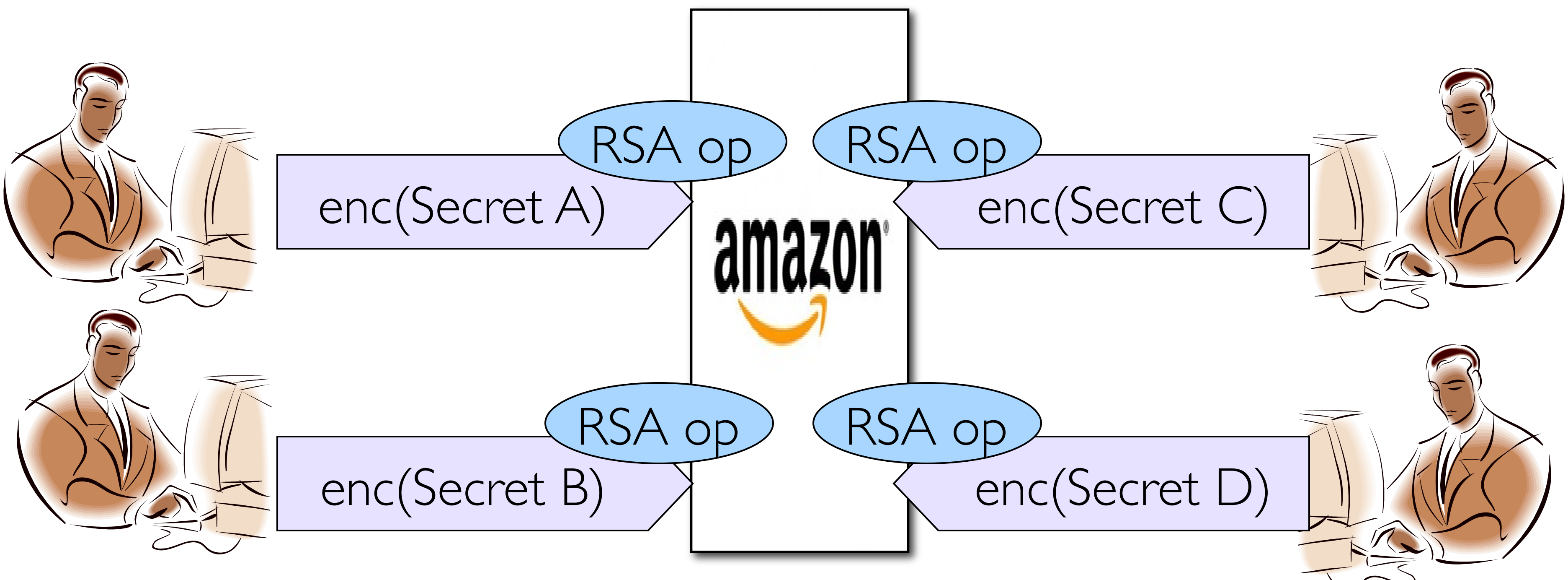
Authentication Example:

Signing a batch of session IDs to amortize RSA costs





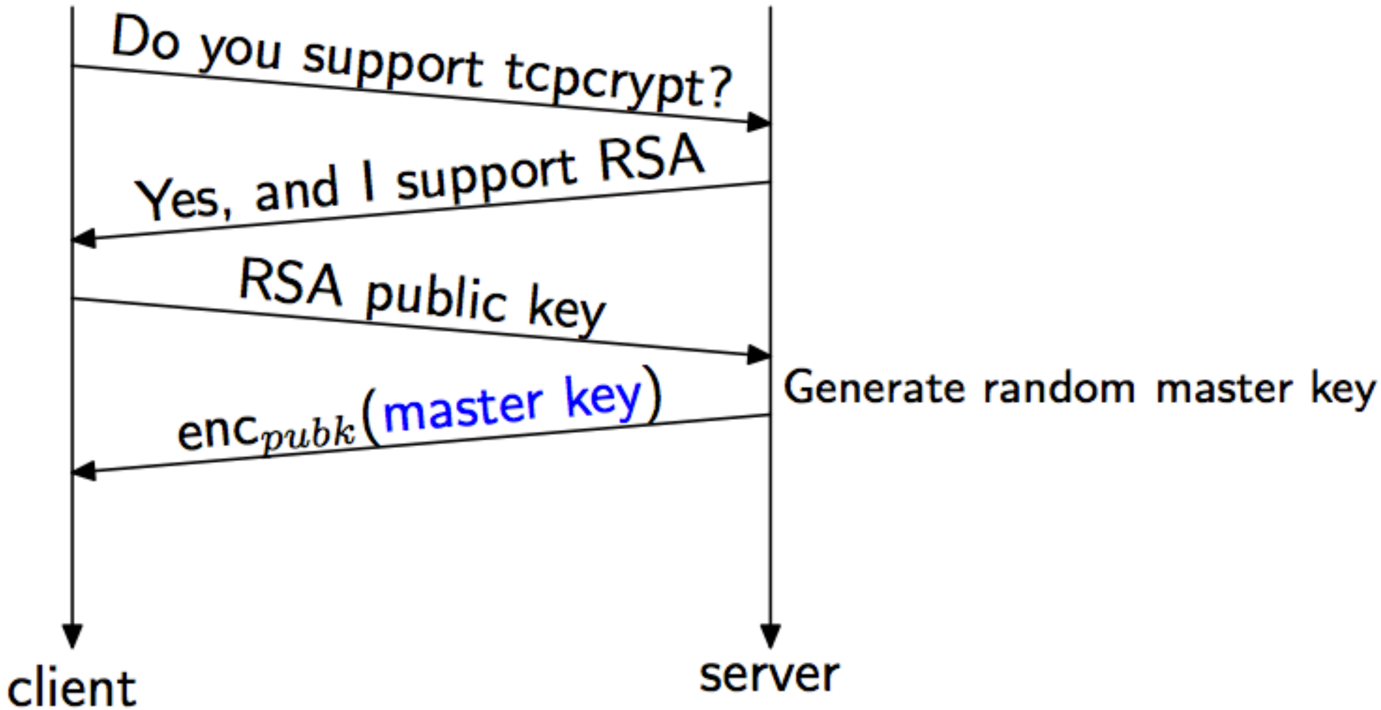
SSL servers RSA decrypt each client's secret:



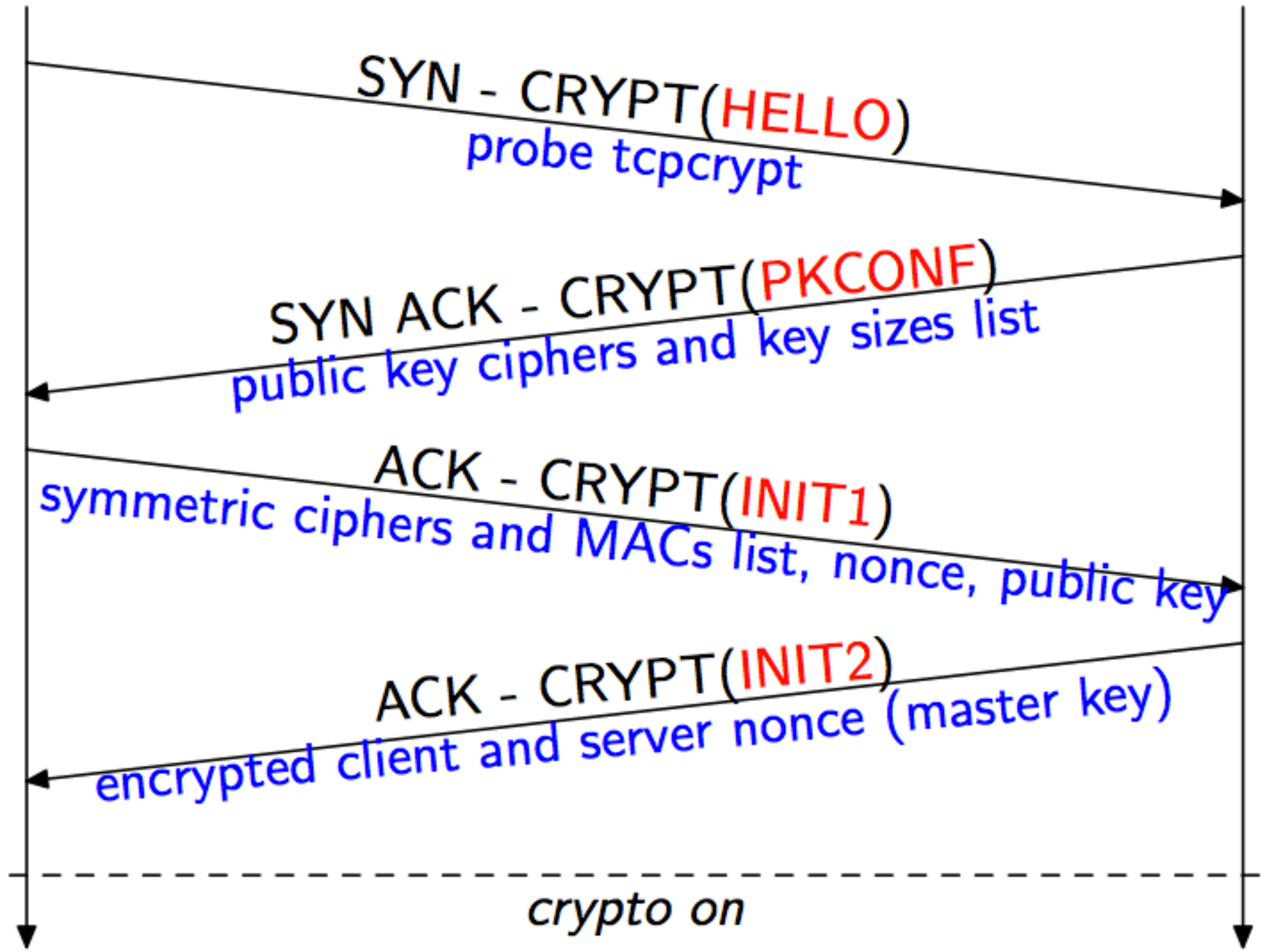
Tcpcrypt

in detail

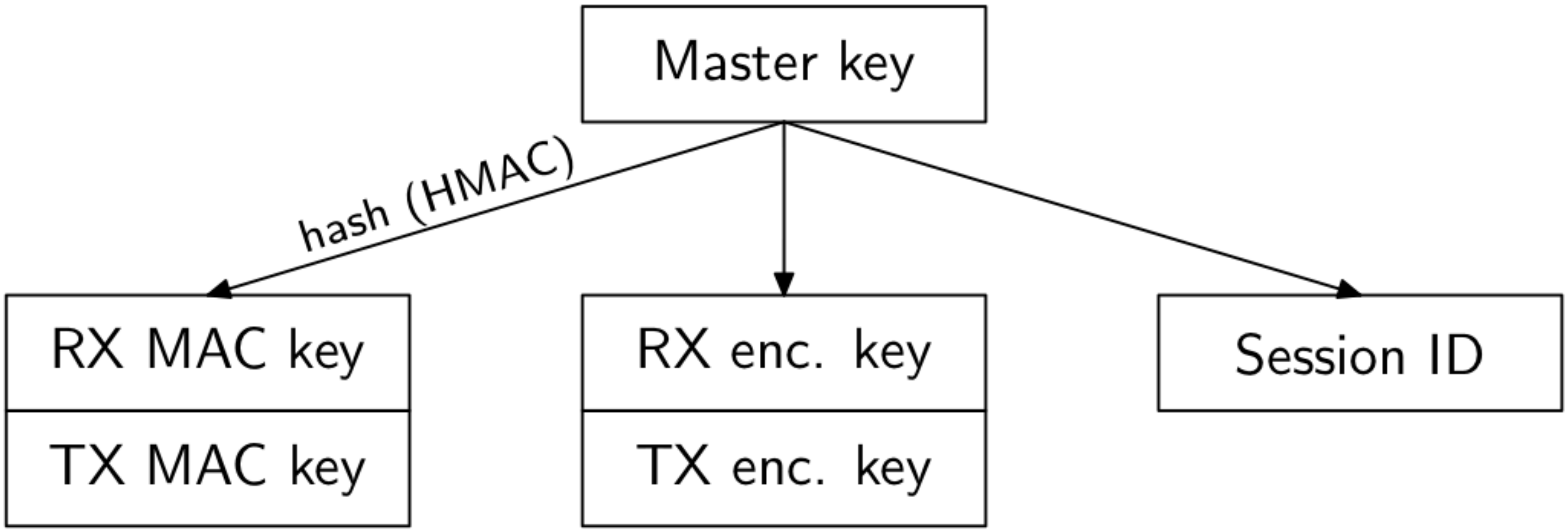
Outline of Tcpcrypt key exchange



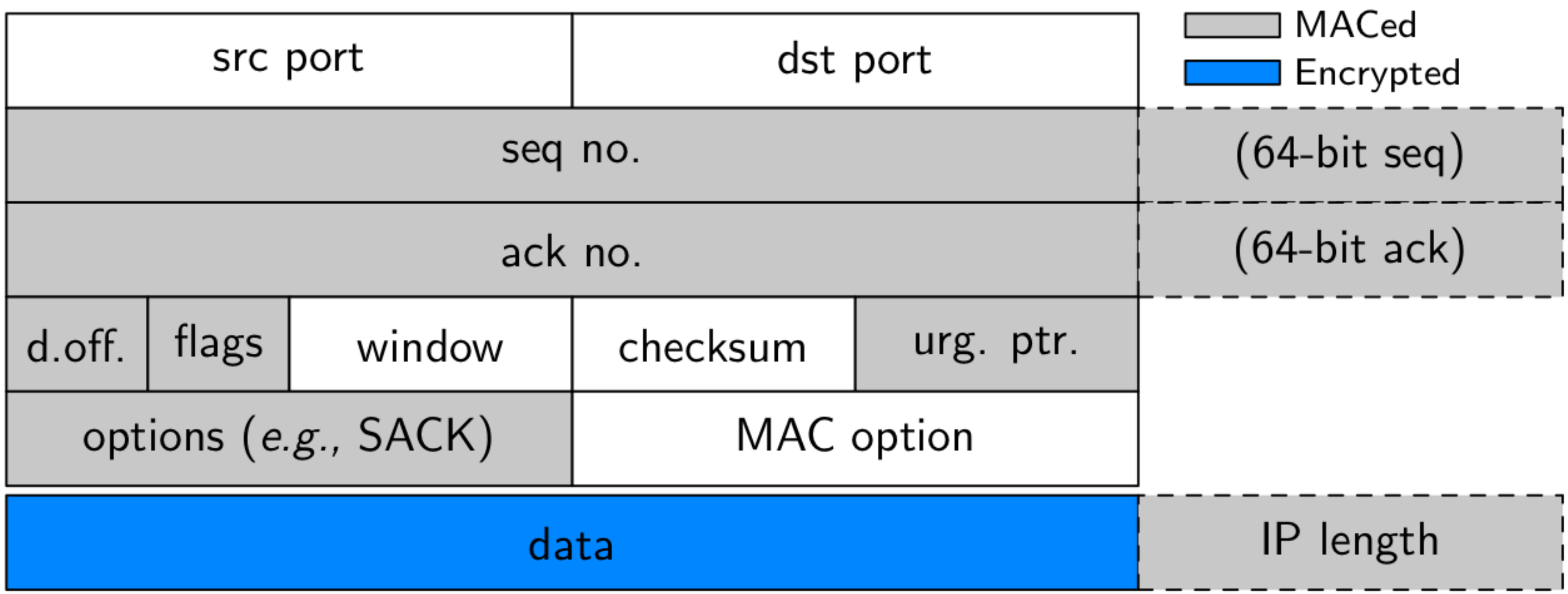
Key exchange is performed in the TCP connection setup handshake.



Key Scheduling

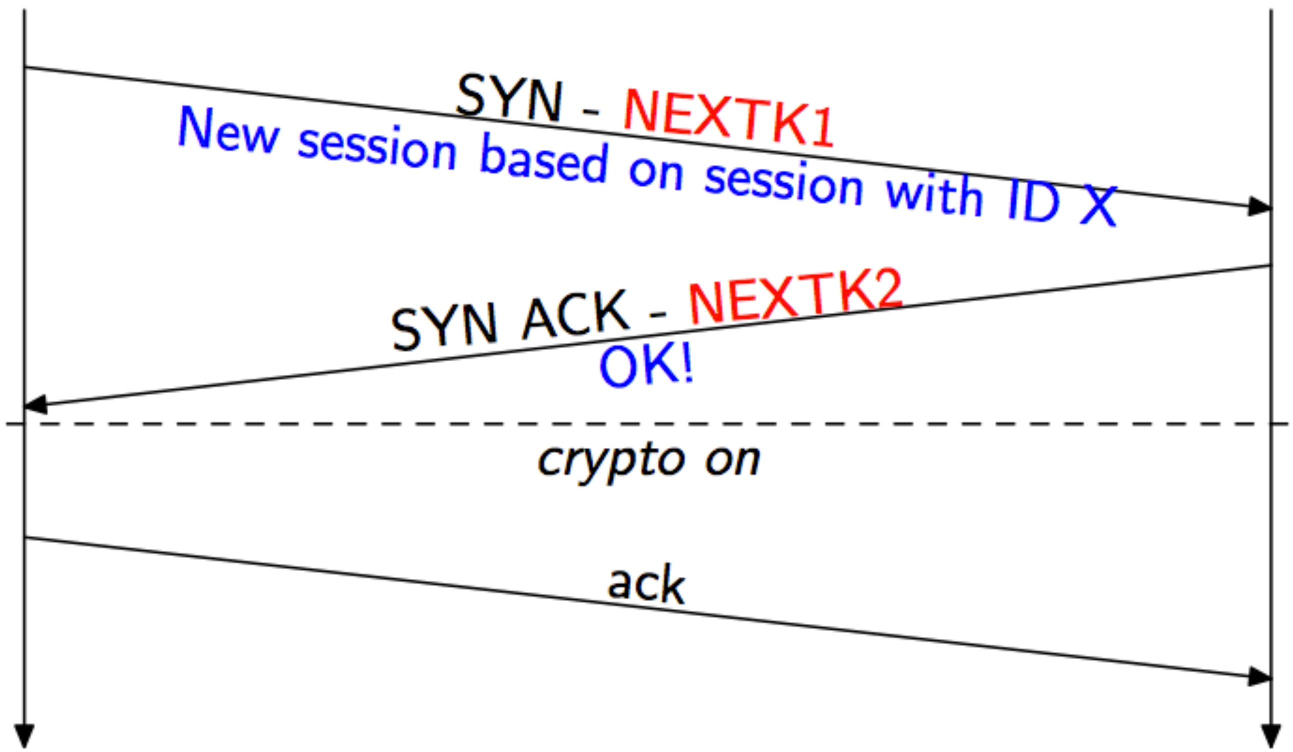


Tcpcrypt in TCP Packets

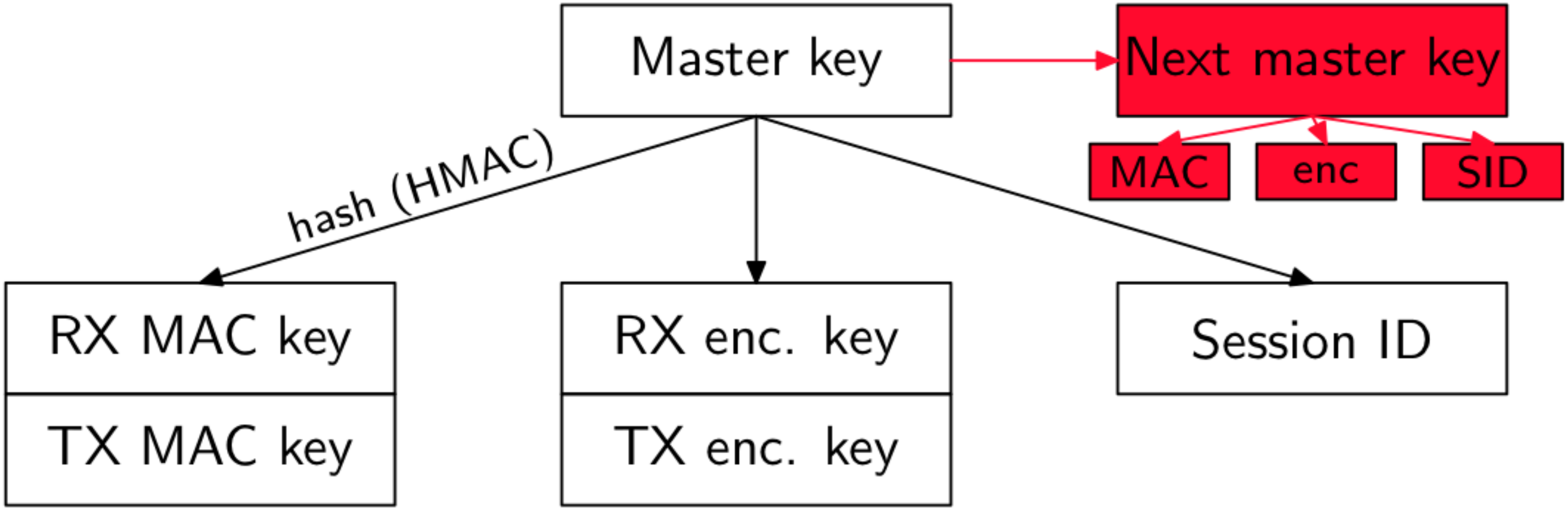


Crypto state can be cached.

Subsequent connections between the same endpoints get similar latency to regular TCP.



Key Scheduling 2

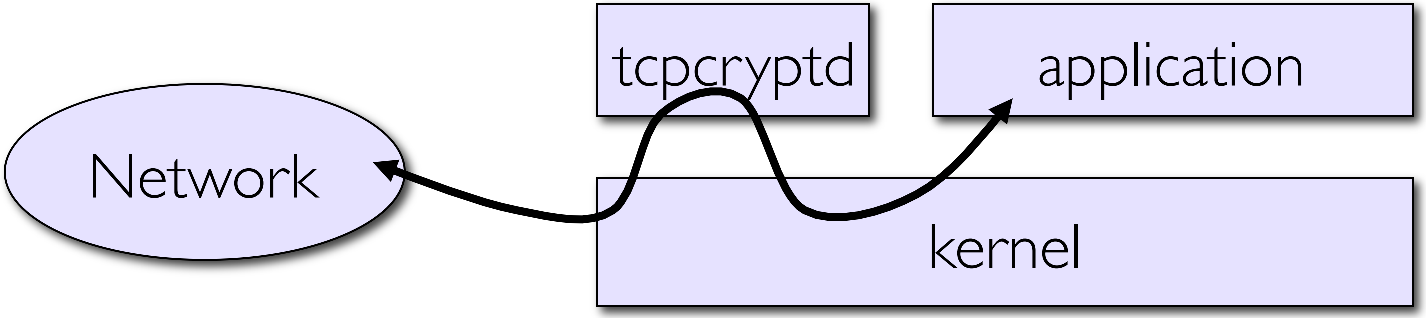


Tcpcrypt

Performance

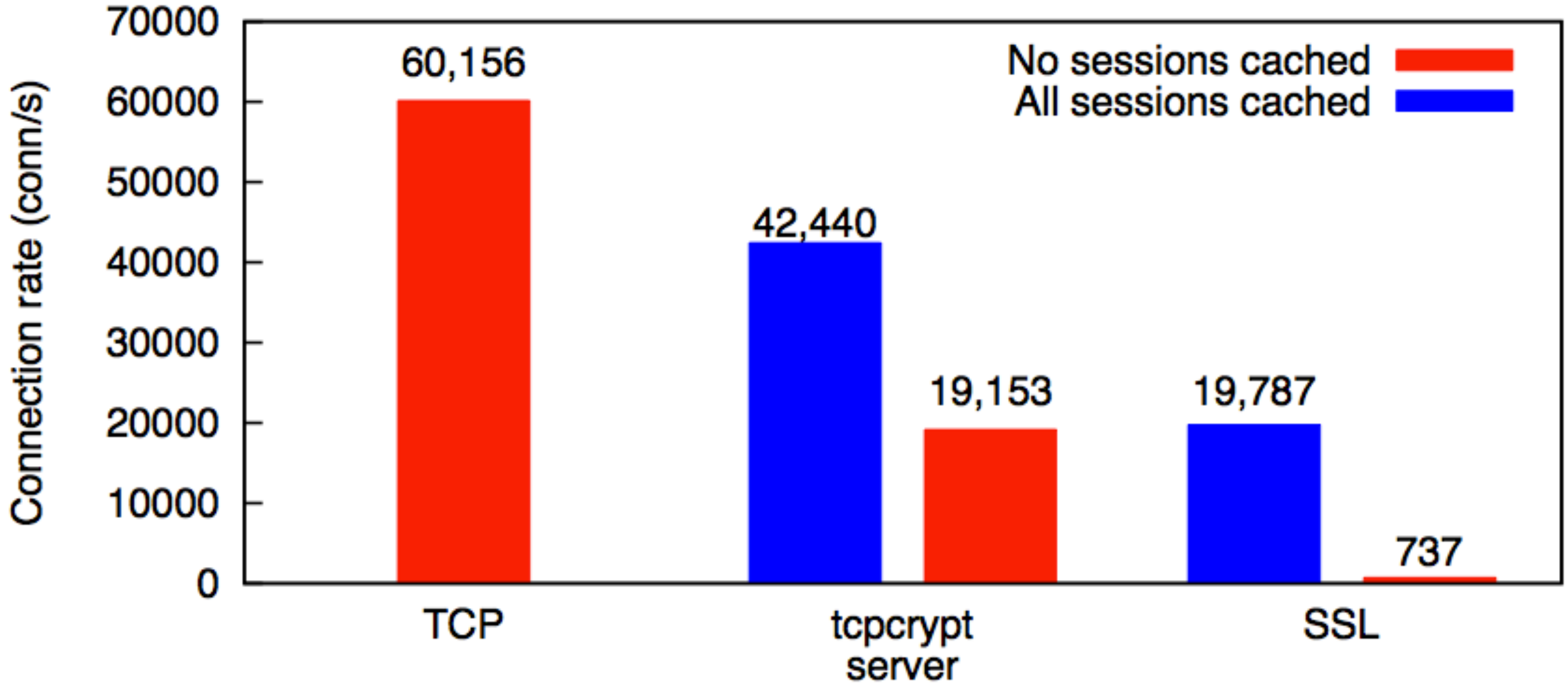
Tcpcrypt implementations

- Linux kernel implementation: 4,500 lines of code
- Portable divert-socket implementation: 7000 LoC
  - Tested on Windows, MacOS, Linux, FreeBSD



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

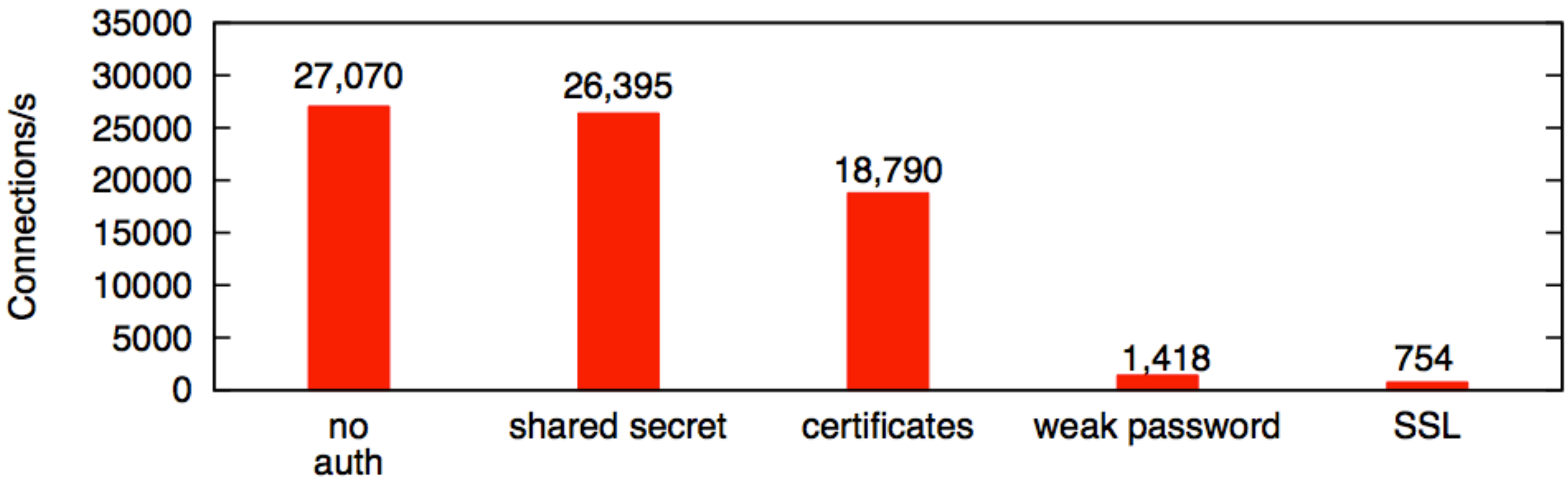
Apache using tcpcrypt performs well.



Hardware: 8-core, 2.66GHz Xeon (2008-era).

Software: Linux kernel implementation.

Authentication over Tcpcrypt is fast.



What would it take to encrypt all the traffic on the Internet, by default, all the time?

Why tcpcrypt?

- Want to protect TCP packet headers.
  - Defend against insertion attacks, etc.
  - But still traverse NATs, firewalls that rewrite sequence numbers.
- Want on-by-default encryption for existing unmodified apps to protect against passive easesdropping.
  - Fast enough to enable on all servers by default.
  - Won't break - downgrades to TCP if necessary.
- Easy incremental deployment story due to negotiation in TCP handshake.

Why tcpcrypt?

- Want to enable and encourage appropriate authentication above tcpcrypt.
  - Cert-based, mutual auth, PAKE, etc, as appropriate.
  - Eg. can support connectbyname() in a shim library, and leverage DANE for auth.

- Separation of layering provides flexibility.
  - Eg. allows corporate firewall to do encryption and app to still do authentication, so corporate IDS still works.
    - tcpcryptd on firewall
    - RPC to get session ID.

Summary: *tcpcrypt* can enable ubiquitous transport level encryption

- High server performance makes encryption a realistic default.
- Applications can leverage Tcpcrypt to maximize communication security in every setting.
- Incrementally deployable, compatible with legacy apps, TCP and NATs.

http://tcpcrypt.org

Spare slides

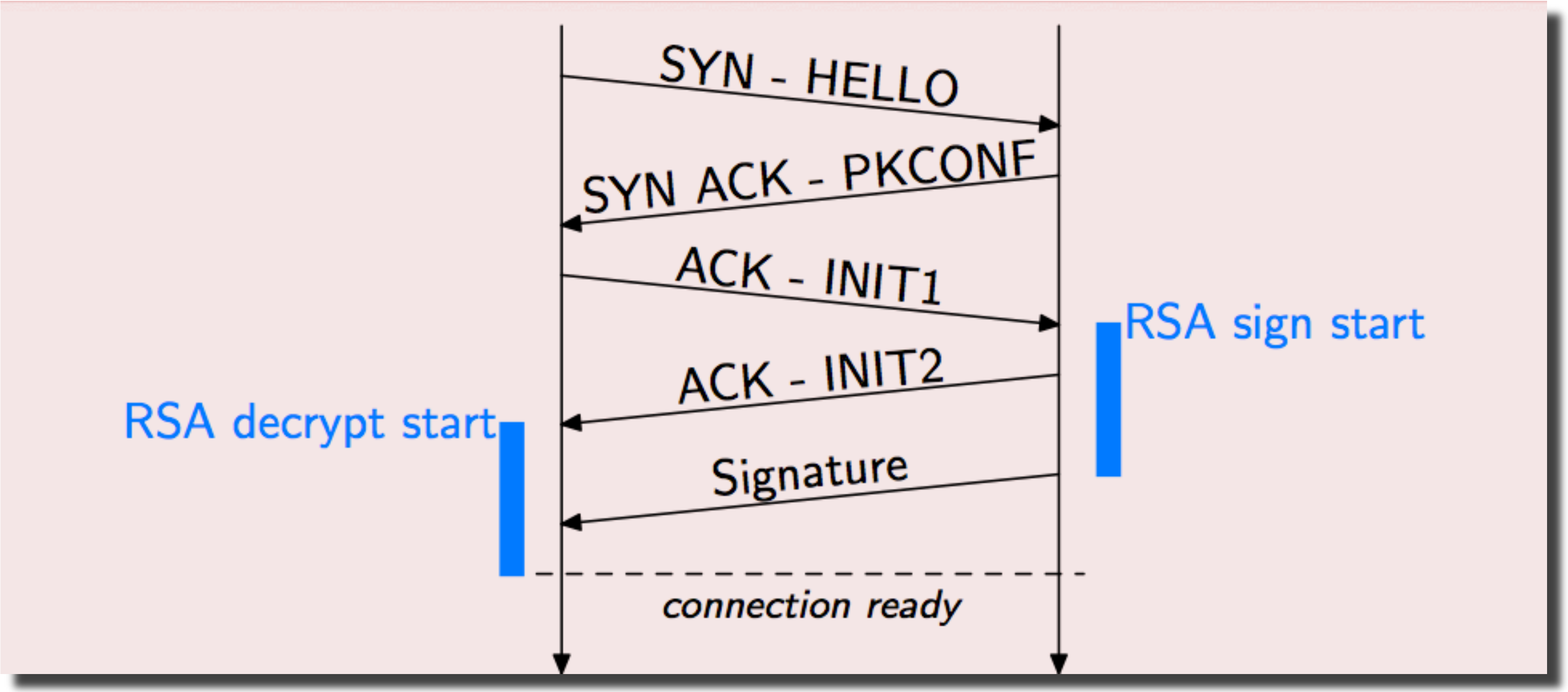
Connection setup latency is slightly increased due to client-side RSA decrypt.

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

Most authentication can be done very cheaply, once the Tcpcrypt session is established.

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

Batch signing does not add additional latency



Data encryption is very fast on today's CPUs