Post-quantum TLS without handshake signatures

Douglas Stebila

UNIVERSITY OF WATERLOO

NSERC CRSNG

https://eprint.iacr.org/2020/534 • https://github.com/thomwiggers/kemtls-experiment/
https://www.douglas.stebila.ca/research/presentations/

Alphabet • 2021-05-13

UNIVERSITY OF WATERLOO

IQC Institute for Quantum Computing

CYBER SECURITY AND PRIVACY INSTITUTE

UNIVERSITY OF WATERLOO

Cryptography @ University of Waterloo

- UW involved in 4 NIST PQC Round 3 submissions:
  - Finalists: CRYSTALS-Kyber, NTRU
  - Alternates: FrodoKEM, SIKE
- UW involved in 4 NIST Lightweight Crypto Round 2 submissions: ACE, SPIX, SpoC, WAGE
- Elliptic curves: David Jao, Alfred Menezes, (Scott Vanstone)
- Information theoretic cryptography: Doug Stinson
- Privacy-enhancing technologies: Ian Goldberg
- Quantum cryptanalysis: Michele Mosca
- Quantum cryptography: Norbert Lütkenhaus, Thomas Jennewein, Debbie Leung
- Gord Agnew, Vijay Ganesh, Guang Gong, Sergey Gorbunov, Anwar Hasan, Florian Kerschbaum

Authenticated key exchange

- Two parties establish a shared secret over a public communication channel

Vast literature on AKE protocols

- Many **security definitions** capturing various adversarial powers: BR, CK, eCK, …
- Different types of **authentication credentials**: public key, shared secret key, password, identity-based, …
- **Additional security goals**: weak/strong forward secrecy, key compromise impersonation resistance, post-compromise security, …
- Additional **protocol functionality**: multi-stage, ratcheting, …
- **Group** key exchange
- **Real-world protocols**: TLS, SSH, Signal, IKE, ISO, EMV, …
- …

Explicit authentication

Alice receives assurance that she really is talking to Bob

Implicit authentication

Alice is assured that only Bob would be able to compute the shared secret

Explicitly authenticated key exchange: Signed Diffie–Hellman

Alice		**Bob**
$(pk_A, sk_A) \leftarrow \text{SIG.KeyGen}()$		$(pk_B, sk_B) \leftarrow \text{SIG.KeyGen}()$
obtain pk_B		obtain pk_A
$x \leftarrow_{\$} \{0, \ldots, q-1\}$		
$X \leftarrow g^x$	X $\longrightarrow$	
		$y \leftarrow_{\$} \{0, \ldots, q-1\}$
		$Y \leftarrow g^y$
	$\longleftarrow$ Y, σ_B	$\sigma_B \leftarrow \text{SIG.Sign}(sk_B, A \| B \| X \| Y)$
$\sigma_A \leftarrow \text{SIG.Sign}(sk_A, A \| B \| X \| Y)$	σ_A $\longrightarrow$	
$k \leftarrow H(sid, Y^x)$		$k \leftarrow H(sid, X^y)$
	application data $\Longleftrightarrow$ using authenticated encryption	

Implicitly authenticated key exchange: Double-DH

Alice		**Bob**
$sk_A \leftarrow_\$ \{0, \ldots, q-1\}$		$sk_B \leftarrow_\$ \{0, \ldots, q-1\}$
$pk_A \leftarrow g^{sk_A}$		$pk_B \leftarrow g^{sk_B}$
obtain pk_B		obtain pk_A
$x \leftarrow_\$ \{0, \ldots, q-1\}$		$y \leftarrow_\$ \{0, \ldots, q-1\}$
$X \leftarrow g^x$	$\xrightarrow{X}$	$Y \leftarrow g^y$
	$\xleftarrow{Y}$	
$k \leftarrow H(sid,\ pk_B^{sk_A} \| Y^x)$		$k \leftarrow H(sid,\ \boxed{pk_A^{sk_B}} \| X^y)$
	application data $\Longleftrightarrow$ using authenticated encryption	

Alphabet

abc.xyz

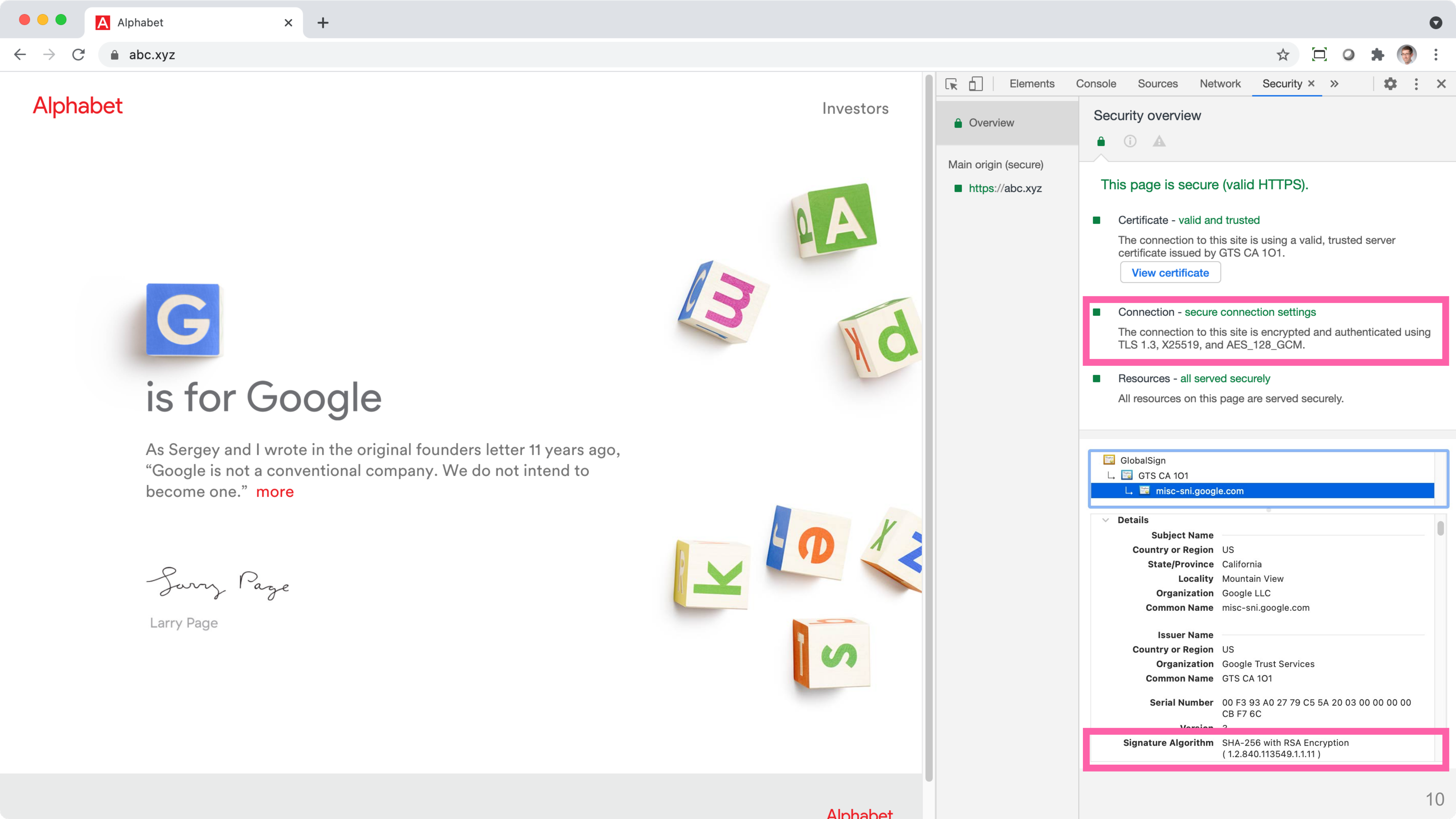
Alphabet

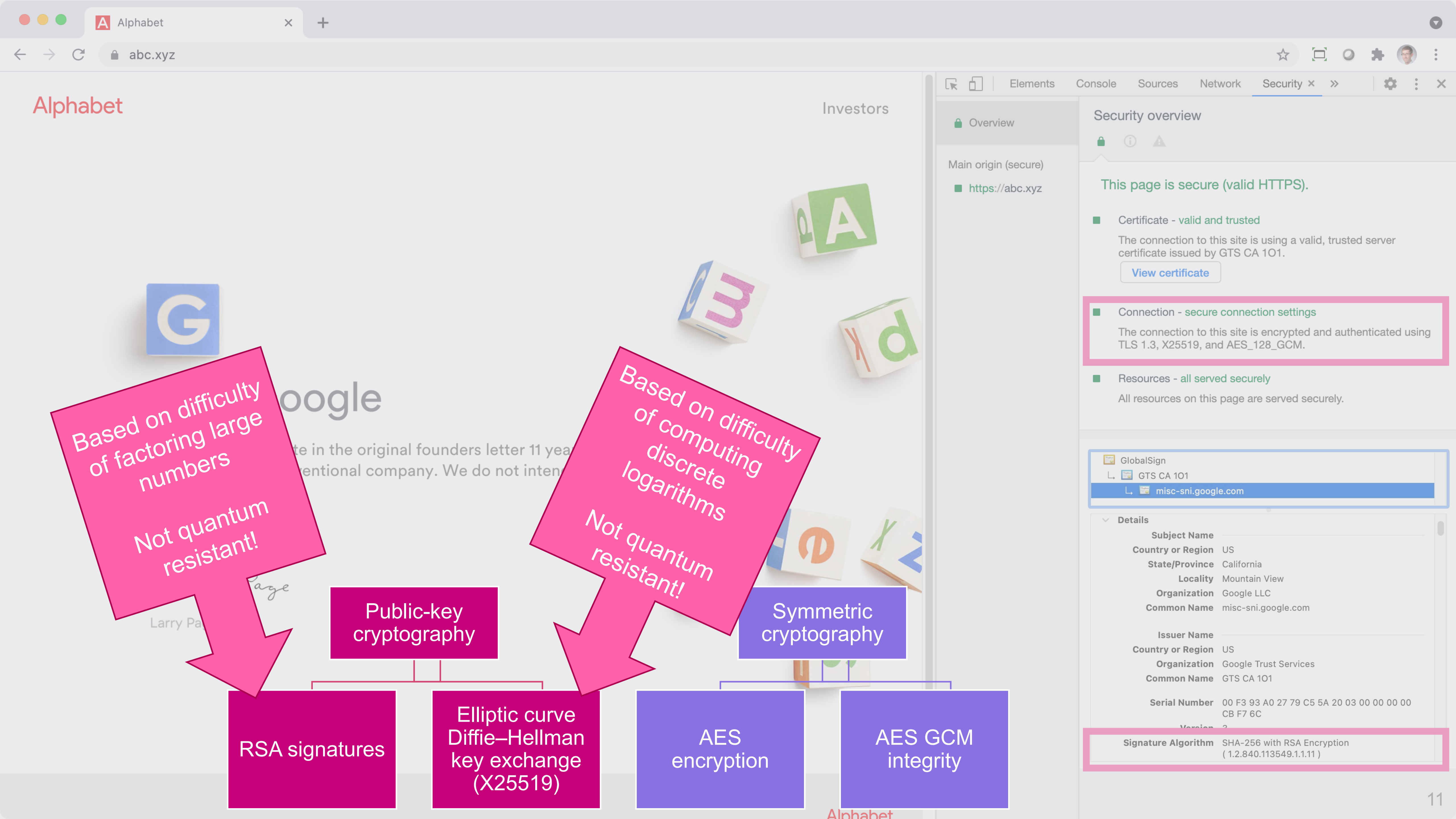
Investors

is for Google

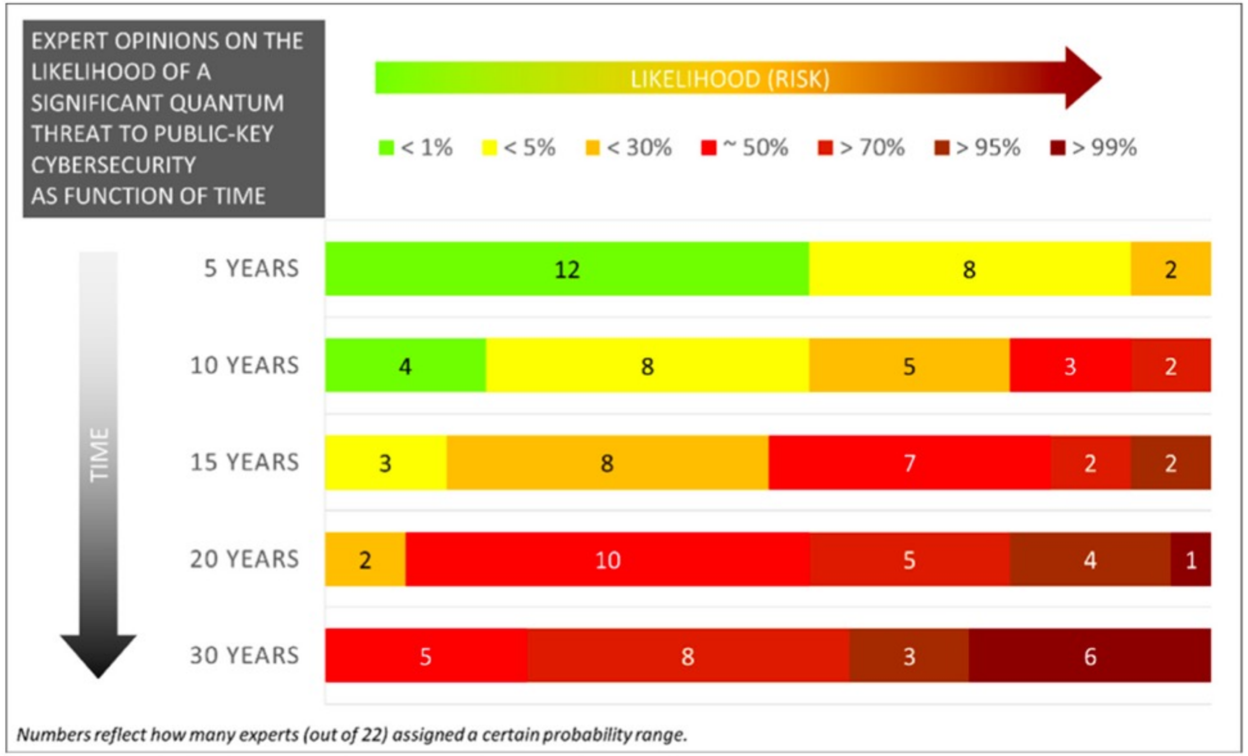
As Sergey and I wrote in the original founders letter 11 years ago, “Google is not a conventional company. We do not intend to become one.” more

Larry Page


Alphabet
abc.xyz
Alphabet
Investors
G
is for Google
As Sergey and I wrote in the original founders letter 11 years ago, “Google is not a conventional company. We do not intend to become one.” more
Larry Page
Elements
Console
Sources
Network
Security
Overview
Main origin (secure)
https://abc.xyz
Security overview
This page is secure (valid HTTPS).
Certificate - valid and trusted
The connection to this site is using a valid, trusted server certificate issued by GTS CA 1O1.
View certificate
Connection - secure connection settings
The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.
Resources - all served securely
All resources on this page are served securely.
GlobalSign
GTS CA 1O1
misc-sni.google.com
Details
Subject Name
Country or Region US
State/Province California
Locality Mountain View
Organization Google LLC
Common Name misc-sni.google.com
Issuer Name
Country or Region US
Organization Google Trust Services
Common Name GTS CA 1O1
Serial Number 00 F3 93 A0 27 79 C5 5A 20 03 00 00 00 00 CB F7 6C
Signature Algorithm SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Alphabet



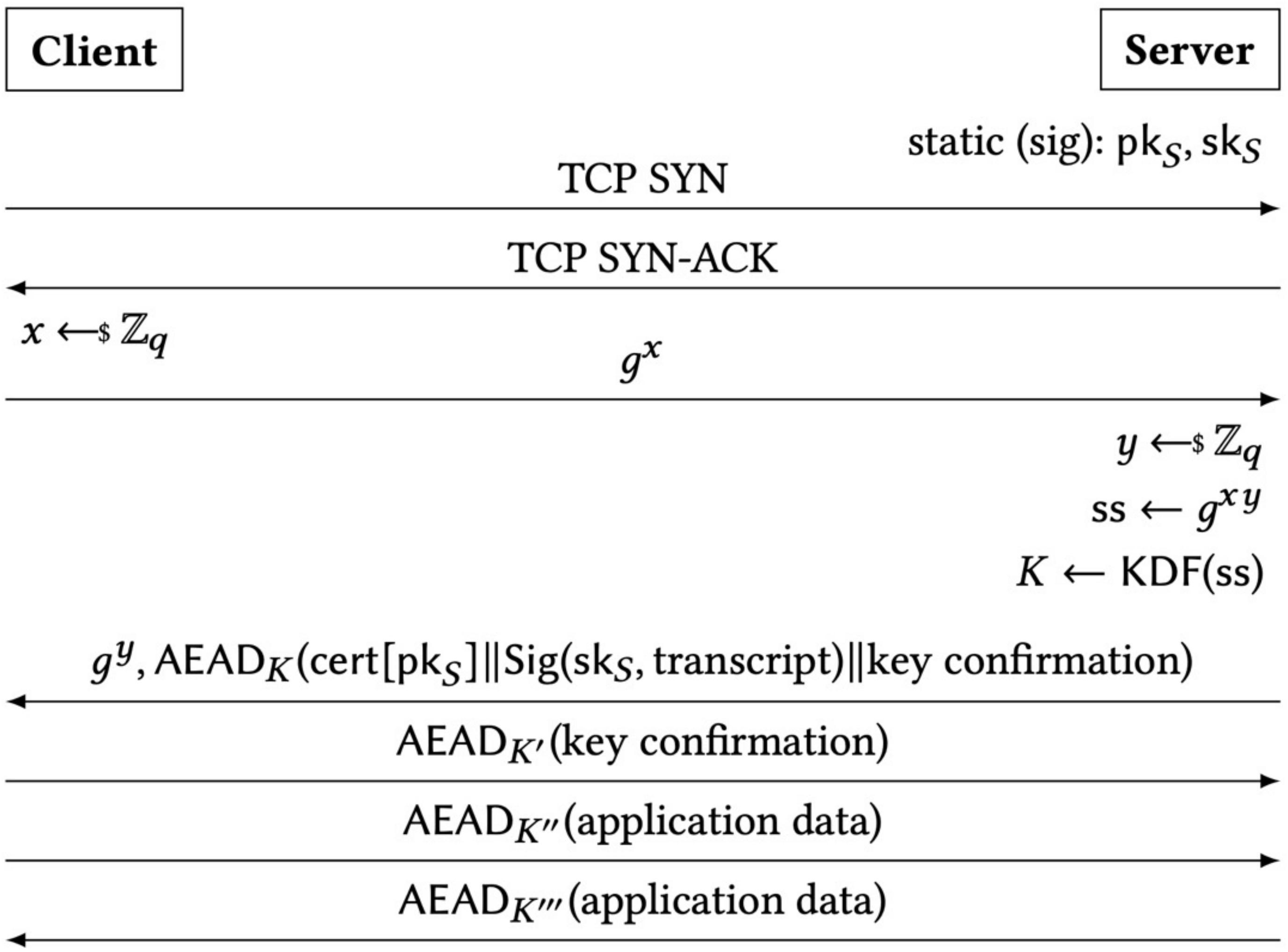
Alphabet
abc.xyz
Elements
Console
Sources
Network
Security
Investors
Overview
Main origin (secure)
https://abc.xyz
Security overview
This page is secure (valid HTTPS).
Certificate - valid and trusted
The connection to this site is using a valid, trusted server certificate issued by GTS CA 1O1.
View certificate
Connection - secure connection settings
The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM.
Resources - all served securely
All resources on this page are served securely.
GlobalSign
GTS CA 1O1
misc-sni.google.com
Details
Subject Name
Country or Region US
State/Province California
Locality Mountain View
Organization Google LLC
Common Name misc-sni.google.com
Issuer Name
Country or Region US
Organization Google Trust Services
Common Name GTS CA 1O1
Serial Number 00 F3 93 A0 27 79 C5 5A 20 03 00 00 00 00 CB F7 6C
Signature Algorithm SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Based on difficulty of factoring large numbers
Not quantum resistant!
Based on difficulty of computing discrete logarithms
Not quantum resistant!
Public-key cryptography
Symmetric cryptography
RSA signatures
Elliptic curve Diffie–Hellman key exchange (X25519)
AES encryption
AES GCM integrity



https://globalriskinstitute.org/publications/quantum-threat-timeline/

TLS 1.3 handshake

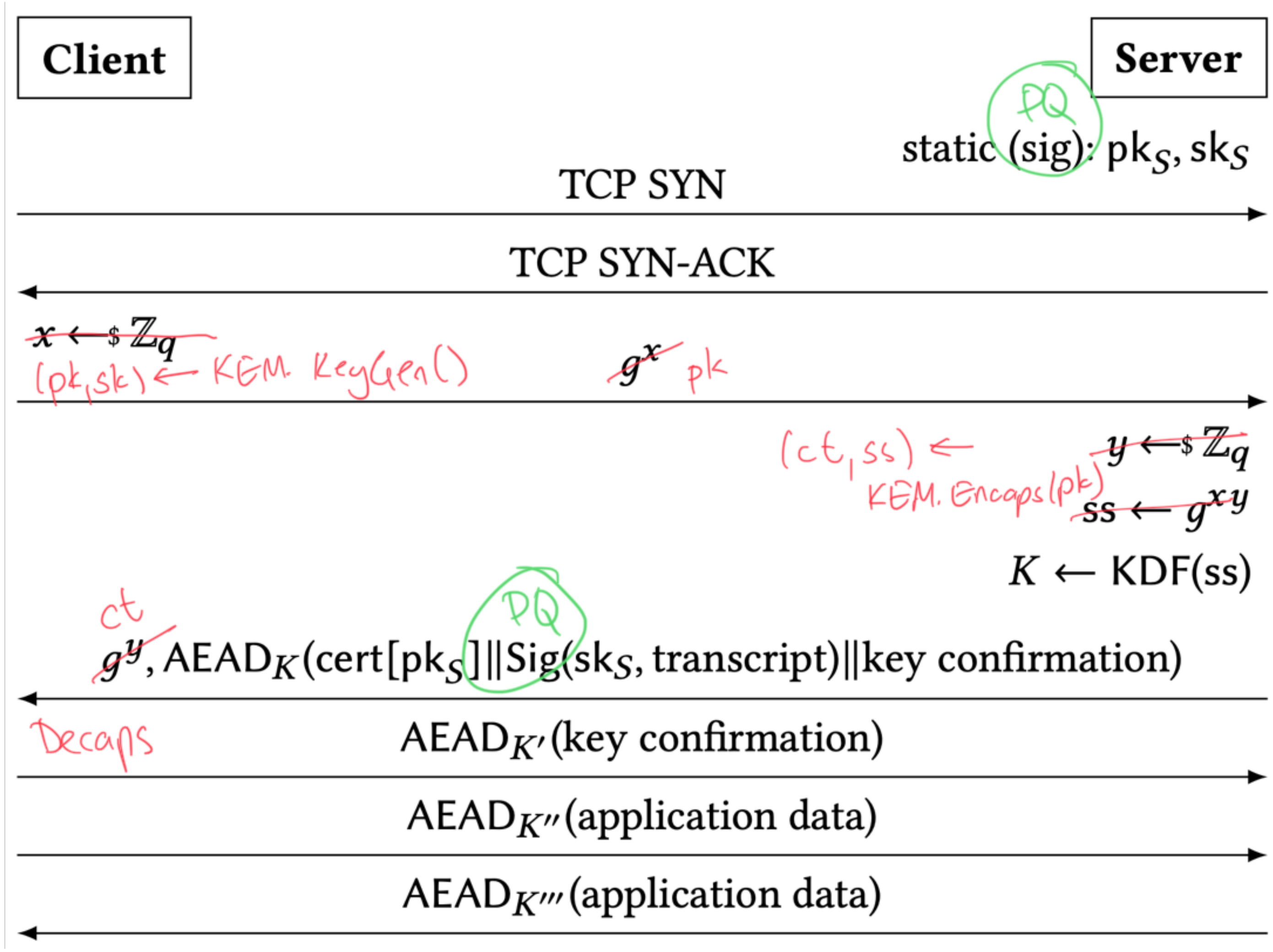
Signed Diffie–Hellman



TLS 1.3 handshake

~~Signed Diffie–Hellman~~

Post-Quantum!!!



Problem

post-quantum
signatures
are big

Signature scheme		Public key (bytes)	Signature (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Dilithium	**Lattice-based (MLWE/MSIS)**	**1,184**	**2,044**
Falcon	**Lattice-based (NTRU)**	**897**	**690**
XMSS	**Hash-based**	**32**	**979**
GeMSS	**Multi-variate**	**352,180**	**32**

Solution

use
post-quantum KEMs
for authentication

Key encapsulation mechanisms (KEMs)

An abstraction of Diffie–Hellman key exchange

$(pk, sk) \leftarrow \mathsf{KEM.KeyGen}()$

$$\xrightarrow{\quad pk \quad}$$

$(ct, k) \leftarrow \mathsf{KEM.Encaps}(pk)$

$$\xleftarrow{\quad ct \quad}$$

$k \leftarrow \mathsf{KEM.Decaps}(sk, ct)$

Signature scheme		Public key (bytes)	Signature (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Dilithium	**Lattice-based (MLWE/MSIS)**	**1,184**	**2,044**
Falcon	**Lattice-based (NTRU)**	**897**	**690**
XMSS	**Hash-based**	**32**	**979**
GeMSS	**Multi-variate**	**352,180**	**32**

KEM		Public key (bytes)	Ciphertext (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
Kyber	**Lattice-based (MLWE)**	**800**	**768**
NTRU	**Lattice-based (NTRU)**	**699**	**699**
Saber	**Lattice-based (MLWR)**	**672**	**736**
SIKE	**Isogeny-based**	**330**	**330**
SIKE compressed	**Isogeny-based**	**197**	**197**
Classic McEliece	**Code-based**	**261,120**	**128**

Implicitly authenticated KEX is not new

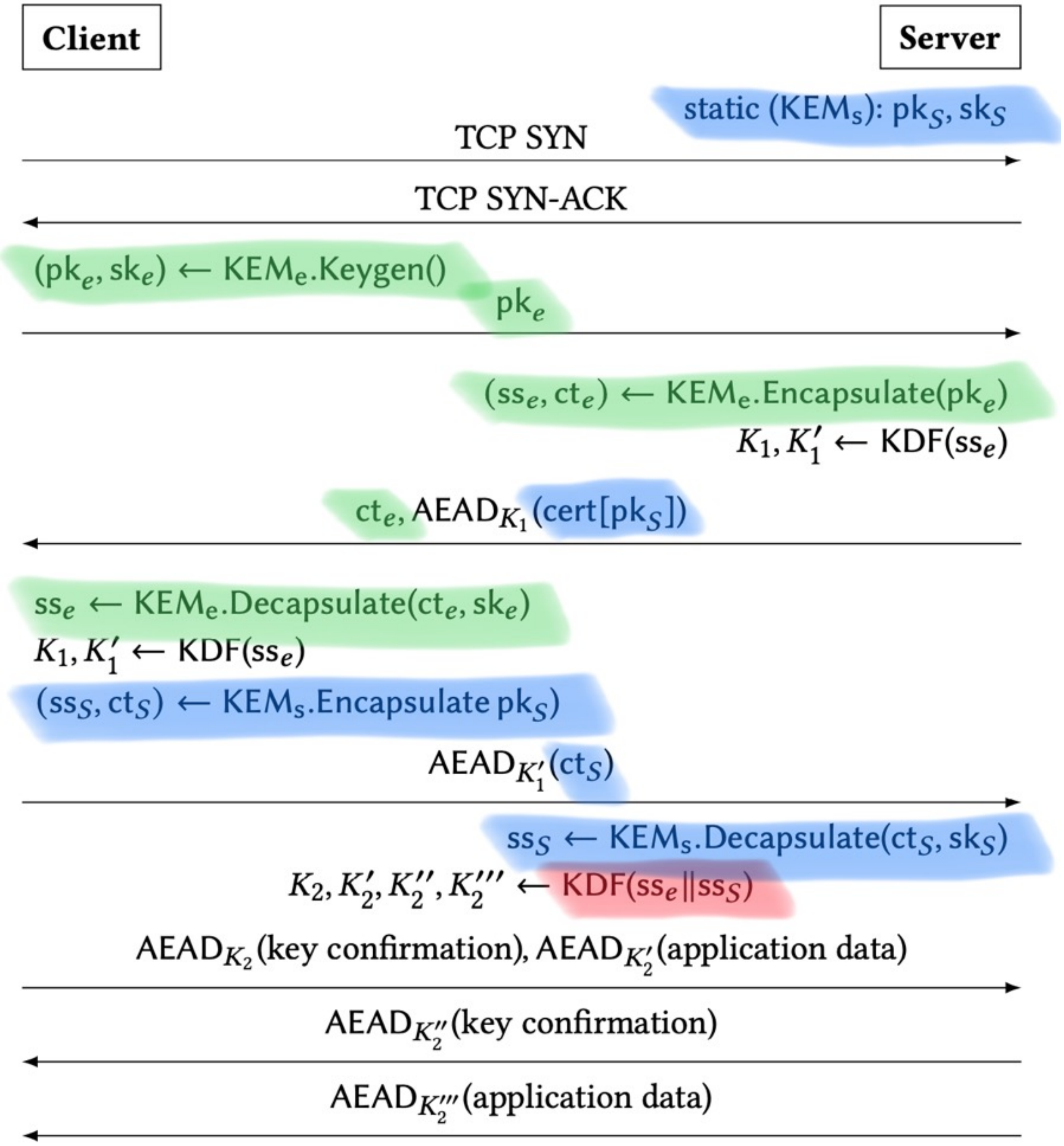
In theory

- DH-based: SKEME, MQV, HMQV, …
- KEM-based: BCGP09, FSXY12, …

In practice

- RSA key transport in TLS ≤ 1.2
  - Lacks forward secrecy
- Signal, Noise, Wireguard
  - DH-based
  - Different protocol flows
- OPTLS
  - DH-based
  - Requires a non-interactive key exchange (NIKE)

“KEMTLS” handshake



KEM for
ephemeral key exchange

KEM for
server-to-client
authenticated key exchange

Combine shared secrets

Algorithm choices

KEM for ephemeral key exchange

- IND-CCA (or IND-1CCA)
- Want small public key + small ciphertext

KEM for authenticated key exchange

- IND-CCA
- Want small public key + small ciphertext

Signature scheme for intermediate CA

- Want small public key + small signature

Signature scheme for root CA

- Want small signature

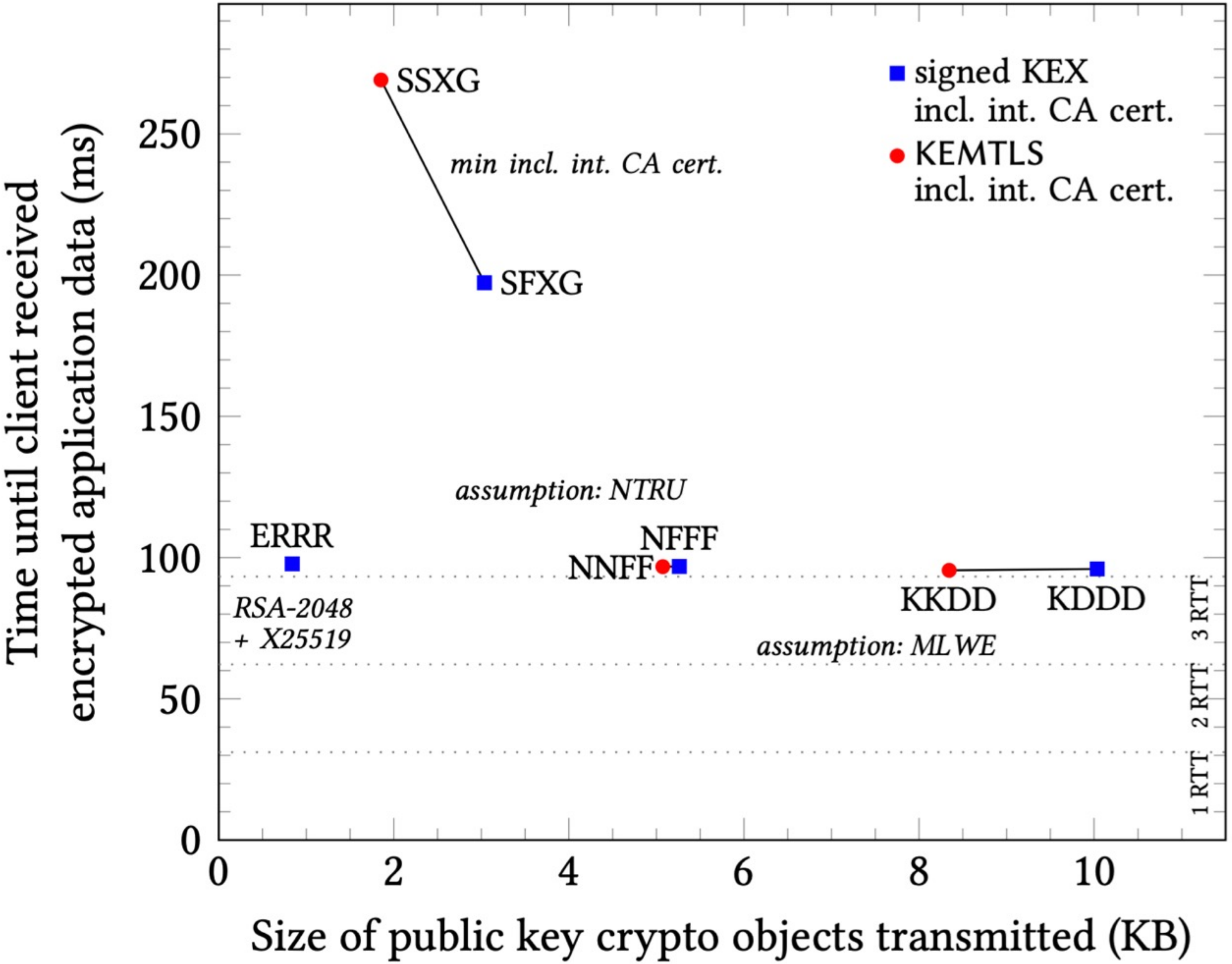
4 scenarios

1. Minimize size when intermediate certificate transmitted
2. Minimize size when intermediate certificate not transmitted (cached)
3. Use solely NTRU assumptions
4. Use solely module LWE/SIS assumptions

Signed KEX versus KEMTLS

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

Algorithms: (all level 1)
Dilithium,
ECDH X25519,
Falcon,
GeMSS,
Kyber,
NTRU,
RSA-2048,
SIKE,
XMSS'

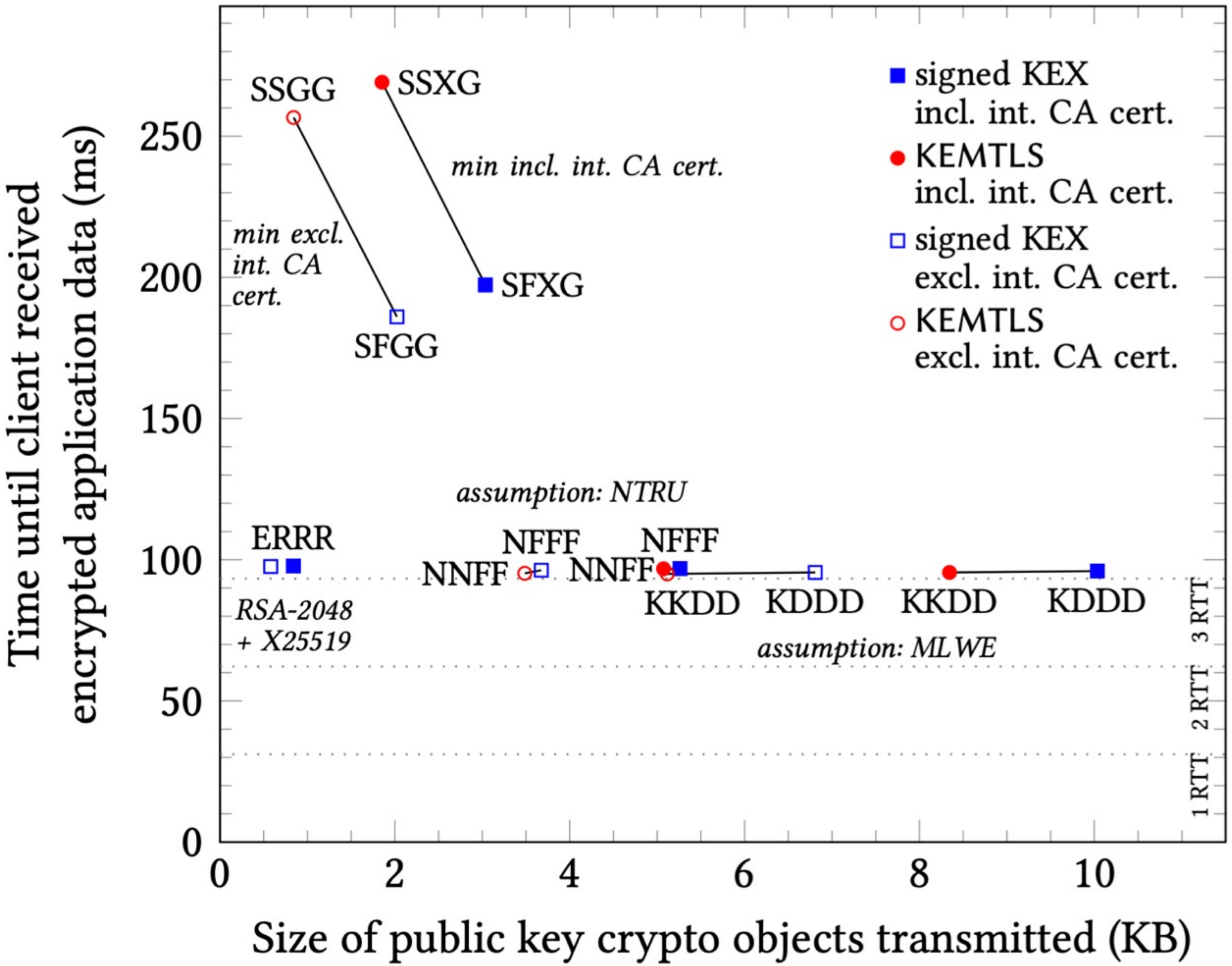


Rustls client/server with some AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

Signed KEX versus KEMTLS

Labels ABCD:
A = ephemeral KEM
B = leaf certificate
C = intermediate CA
D = root CA

Algorithms: (all level 1)
Dilithium,
ECDH X25519,
Falcon,
GeMSS,
Kyber,
NTRU,
RSA-2048,
SIKE,
XMSS'



Rustls client/server with some AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

KEMTLS benefits

- Size-optimized KEMTLS requires < ½ communication of size-optimized PQ signed-KEM
- Speed-optimized KEMTLS uses 90% fewer server CPU cycles and still reduces communication
  - NTRU KEX (27 μs) 10x faster than Falcon signing (254 μs)
- No extra round trips required until client starts sending application data
- Smaller trusted code base (no signature generation on client/server)

Security

Security model: multi-stage key exchange, extending [DFGS21]

- Key indistinguishability
- Forward secrecy
- Implicit and explicit authentication

Ingredients in security proof:

- **IND-CCA for long-term KEM**
- **IND-1CCA for ephemeral KEM**
- Collision-resistant hash function
- Dual-PRF security of HKDF
- EUF-CMA of HMAC

[DFGS21] Dowling, Fischlin, Günther, Stebila. Journal of Cryptology, 2021. https://eprint.iacr.org/2020/1044

Security subtleties: authentication

Implicit authentication

- Client's first application flow can't be read by anyone other than intended server, but client doesn't know server is live at the time of sending

Explicit authentication

- Explicit authentication once key confirmation message transmitted
- *Retroactive* explicit authentication of earlier keys

[DGK06] Di Raimondo, Gennaro, Krawczyk. ACM CCS 2006. https://eprint.iacr.org/2006/280

Security subtleties: downgrade resilience

- Choice of cryptographic algorithms not authenticated at the time the client sends its first application flow
  - MITM can't trick client into using undesirable algorithm
  - But MITM *can* trick them into *temporarily* using suboptimal algorithm

- Formally model 3 levels of downgrade-resilience:
  1. Full downgrade resilience
  2. No downgrade resilience to unsupported algorithms
  3. No downgrade resilience

Security subtleties: forward secrecy

Does compromise of a party’s long-term key allow decryption of past sessions?

- **Weak forward secrecy 1:** adversary passive in the test stage
- **Weak forward secrecy 2:** adversary passive in the test stage or never corrupted peer’s long-term key
- **Forward secrecy:** adversary passive in the test stage or didn’t corrupt peer’s long-term key before acceptance

Variant: KEMTLS with client authentication

1. Client has a long-term KEM public key
2. Client transmits it encrypted under key derived from
   a) server long-term KEM key exchange
   b) ephemeral KEM key exchange

- Adds extra round trip

Variant: Pre-distributed public keys

What if server public keys are pre-distributed?

- Cached in a browser
- Pinned in mobile apps
- Embedded in IoT devices
- Out-of-band (e.g., DNS)
- TLS 1.3: RFC 7924

TLS 1.3 already supports pre-shared symmetric keys

- Harder(?) key management problem
- Different compromise model

KEMTLS-PDK

- Alternate KEMTLS protocol flow when server certificates are known in advance

https://thomwiggers.nl/publication/kemtlspdk/

KEMTLS-PDK benefits

- Additional bandwidth savings
- Makes some PQ algorithms viable
  - Large public keys, small ciphertexts/signatures: Classic McEliece and Rainbow
- Client authentication 1 round-trip earlier if proactive
- Explicit server authentication 1 round-trip earlier
  - => better downgrade resilience

	KEMTLS	Cached TLS	KEMTLS-PDK
Unilaterally authenticated			
Round trips until client receives response data	3	3	3
Size (bytes) of public key crypto objects transmitted:			
• Minimum PQ	932	499	561
• Module-LWE/Module-SIS (Kyber, Dilithium)	5,556	3,988	2,336
• NTRU-based (NTRU, Falcon)	3,486	2,088	2,144
Mutually authenticated			
Round trips until client receives response data	4	3	3
Size (bytes) of public key crypto objects transmitted:			
• Minimum PQ	1,431	2,152	1,060
• MLWE/MSIS	9,554	10,140	6,324
• NTRU	5,574	4,365	4,185

Other security properties

Anonymity

- Client certificate encrypted
- Server certificate encrypted
- Server identity not protected
  - Due to Server Name Indication extension
  - May be able to combine KEMTLS-PDK with Encrypted ClientHello?

Deniability

- KEMTLS and KEMTLS-PDK don’t use signatures for authentication
- Yields offline deniability
  - Judge cannot distinguish honest transcript from forgery
- Does not yield online deniability
  - When one party doesn’t follow protocol or colludes with jduge

TLS ecosystem is complex – lots to consider!

- Datagram TLS
- Use of TLS handshake in other protocols
  - e.g. QUIC
- Application-specific behaviour
  - e.g. HTTP3 SETTINGS frame not server authenticated
- PKI involving KEM public keys
- Long tail of implementations
- …

X.509 certificates for KEM public keys: Proof of possession

Starting to be discussed on IETF LAMPS mailing list and part of re-charter [1,2]

How does requester prove possession of corresponding secret keys?

- Interactive challenge-response protocol: RFC 4210 Sect. 5.2.8.3
- Send certificate back encrypted under subject public key RFC 4210 Sect. 5.2.8.2
  - Weird confidentiality requirement on certificate. Maybe broken by Certificate Transparency?
- Non-interactive certificate signing requests: Not a signature scheme!
  - Research in progress: Can build a not-too-inefficient Picnic-like signature scheme from the KEM operation

Thanks to Mike Ounsworth (Entrust Datacard) for raising some of these issues.

[1] https://mailarchive.ietf.org/arch/msg/spasm/FCCZv3Xi3rkbZyZWQnnMQM0EFYY/
[2] https://mailarchive.ietf.org/arch/msg/spasm/9tukY1yTOzuNE8yHhuBLxzQWAko/

X.509 certificates for KEM public keys: Revocation

How can a certificate owner authorize a revocation request?

- Interactive?
- Use a second signature public key?
- Zero knowledge proof to transform into a signature scheme?

Post-quantum TLS without handshake signatures

Douglas Stebila
University of Waterloo

KEMTLS

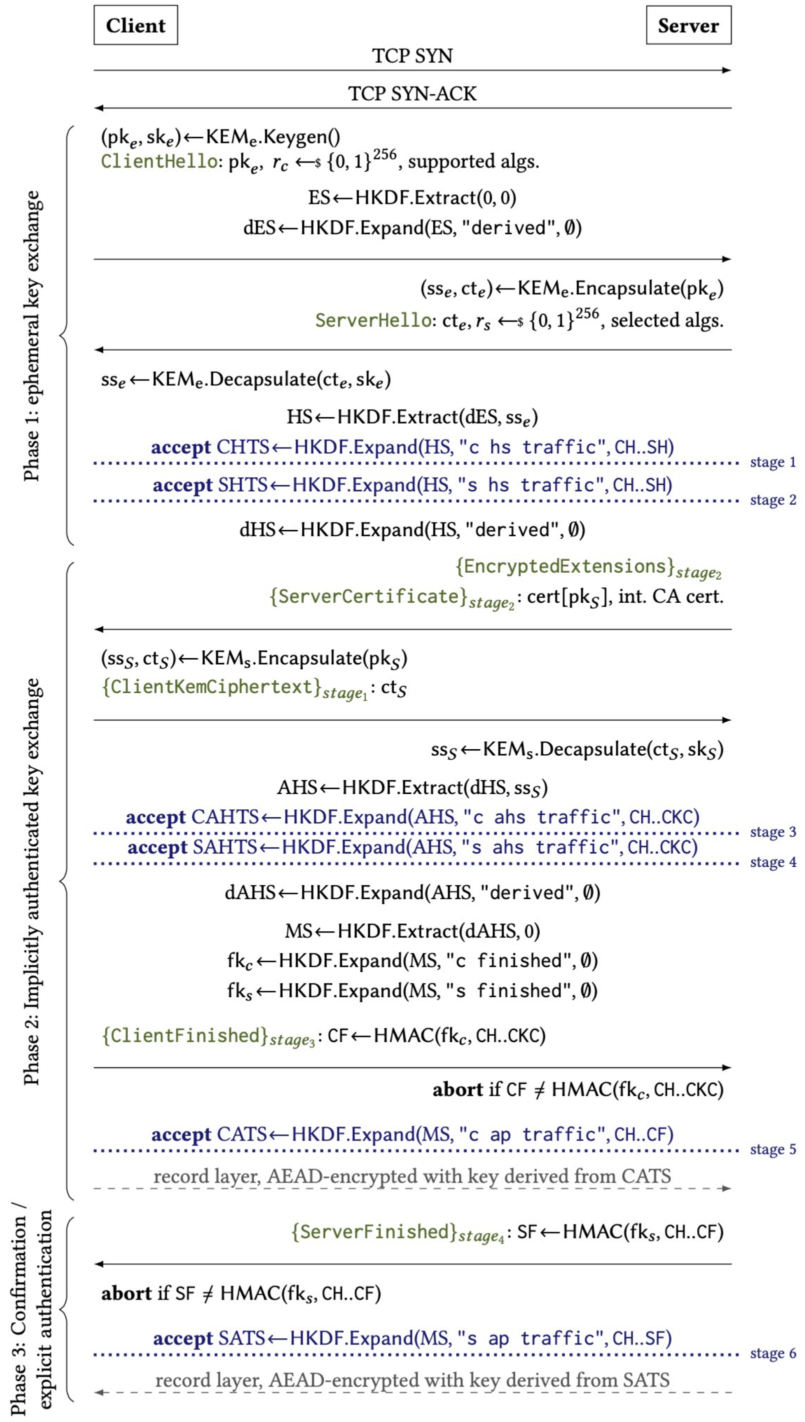
Implicitly authenticated TLS without handshake signatures using KEMs

- Saves bytes on the wire and server CPU cycles
- Preserves client request after 1-RTT
- Caching intermediate CA certs brings even greater benefits

- Variants for client authentication and pre-distributed public keys
- Simple to implement
  - Demos in Rustls, BoringSSL
- Lots of work to make viable in TLS ecosystem, including PKI
- Working with Cloudflare to test within their infrastructure

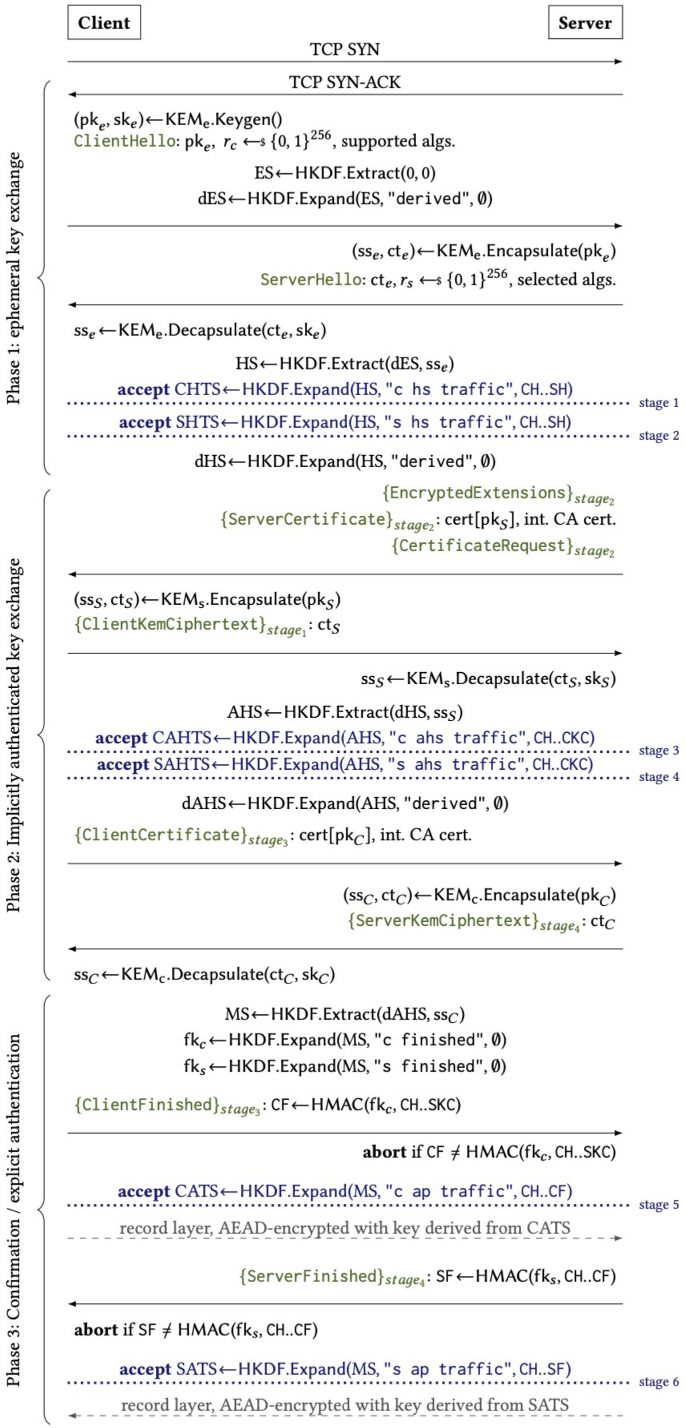
https://eprint.iacr.org/2020/534 • https://github.com/thomwiggers/kemtls-experiment/
https://www.douglas.stebila.ca/research/presentations/

KEMTLS



Client
Server
TCP SYN
TCP SYN-ACK
Phase 1: ephemeral key exchange
(pk_e, sk_e)←KEM_e.Keygen()
ClientHello: pk_e, r_c ←$ {0, 1}^256, supported algs.
ES←HKDF.Extract(0, 0)
dES←HKDF.Expand(ES, "derived", ∅)
(ss_e, ct_e)←KEM_e.Encapsulate(pk_e)
ServerHello: ct_e, r_s ←$ {0, 1}^256, selected algs.
ss_e←KEM_e.Decapsulate(ct_e, sk_e)
HS←HKDF.Extract(dES, ss_e)
accept CHTS←HKDF.Expand(HS, "c hs traffic", CH..SH)
stage 1
accept SHTS←HKDF.Expand(HS, "s hs traffic", CH..SH)
stage 2
dHS←HKDF.Expand(HS, "derived", ∅)
Phase 2: Implicitly authenticated key exchange
{EncryptedExtensions}stage2
{ServerCertificate}stage2: cert[pk_S], int. CA cert.
(ss_S, ct_S)←KEM_s.Encapsulate(pk_S)
{ClientKemCiphertext}stage1: ct_S
ss_S←KEM_s.Decapsulate(ct_S, sk_S)
AHS←HKDF.Extract(dHS, ss_S)
accept CAHTS←HKDF.Expand(AHS, "c ahs traffic", CH..CKC)
stage 3
accept SAHTS←HKDF.Expand(AHS, "s ahs traffic", CH..CKC)
stage 4
dAHS←HKDF.Expand(AHS, "derived", ∅)
MS←HKDF.Extract(dAHS, 0)
fk_c←HKDF.Expand(MS, "c finished", ∅)
fk_s←HKDF.Expand(MS, "s finished", ∅)
{ClientFinished}stage3: CF←HMAC(fk_c, CH..CKC)
abort if CF ≠ HMAC(fk_c, CH..CKC)
accept CATS←HKDF.Expand(MS, "c ap traffic", CH..CF)
stage 5
record layer, AEAD-encrypted with key derived from CATS
Phase 3: Confirmation / explicit authentication
{ServerFinished}stage4: SF←HMAC(fk_s, CH..CF)
abort if SF ≠ HMAC(fk_s, CH..CF)
accept SATS←HKDF.Expand(MS, "s ap traffic", CH..SF)
stage 6
record layer, AEAD-encrypted with key derived from SATS

KEMTLS with client authentication



TLS 1.3 and KEMTLS size of public key objects

		Abbrv.	KEX (pk+ct)	Excluding intermediate CA certificate: HS auth (ct/sig)	Leaf crt. subject (pk)	Leaf crt. (signature)	Sum excl. int. CA cert.	Including intermediate CA certificate: Int. CA crt. subject (pk)	Int. CA crt. (signature)	Sum incl. int. CA crt.	Root CA (pk)	Sum TCP payloads of TLS HS (incl. int. CA crt.)
	TLS 1.3	ERRR	ECDH (X25519) 64	RSA-2048 256	RSA-2048 272	RSA-2048 256	**848**	RSA-2048 272	RSA-2048 256	**1376**	RSA-2048 272	2711
TLS 1.3 (Signed KEX)	**Min. incl. int. CA cert.**	SFXG	SIKE 405	Falcon 690	Falcon 897	$XMSS_s^{MT}$ 979	**2971**	$XMSS_s^{MT}$ 32	GeMSS 32	**3035**	GeMSS 352180	4056
TLS 1.3 (Signed KEX)	**Min. excl. int. CA cert.**	SFGG	SIKE 405	Falcon 690	Falcon 897	GeMSS 32	**2024**	GeMSS 352180	GeMSS 32	**354236**	GeMSS 352180	355737
TLS 1.3 (Signed KEX)	**Assumption: MLWE+MSIS**	KDDD	Kyber 1536	Dilithium 2044	Dilithium 1184	Dilithium 2044	**6808**	Dilithium 1184	Dilithium 2044	**10036**	Dilithium 1184	11094
TLS 1.3 (Signed KEX)	**Assumption: NTRU**	NFFF	NTRU 1398	Falcon 690	Falcon 897	Falcon 690	**3675**	Falcon 897	Falcon 690	**5262**	Falcon 897	6227
KEMTLS	**Min. incl. int. CA cert.**	SSXG	SIKE 405	SIKE 209	SIKE 196	$XMSS_s^{MT}$ 979	**1789**	$XMSS_s^{MT}$ 32	GeMSS 32	**1853**	GeMSS 352180	2898
KEMTLS	**Min. excl. int. CA cert.**	SSGG	SIKE 405	SIKE 209	SIKE 196	GeMSS 32	**842**	GeMSS 352180	GeMSS 32	**353054**	GeMSS 352180	354578
KEMTLS	**Assumption: MLWE+MSIS**	KKDD	Kyber 1536	Kyber 736	Kyber 800	Dilithium 2044	**5116**	Dilithium 1184	Dilithium 2044	**8344**	Dilithium 1184	9398
KEMTLS	**Assumption: NTRU**	NNFF	NTRU 1398	NTRU 699	NTRU 699	Falcon 690	**3486**	Falcon 897	Falcon 690	**5073**	Falcon 897	6066

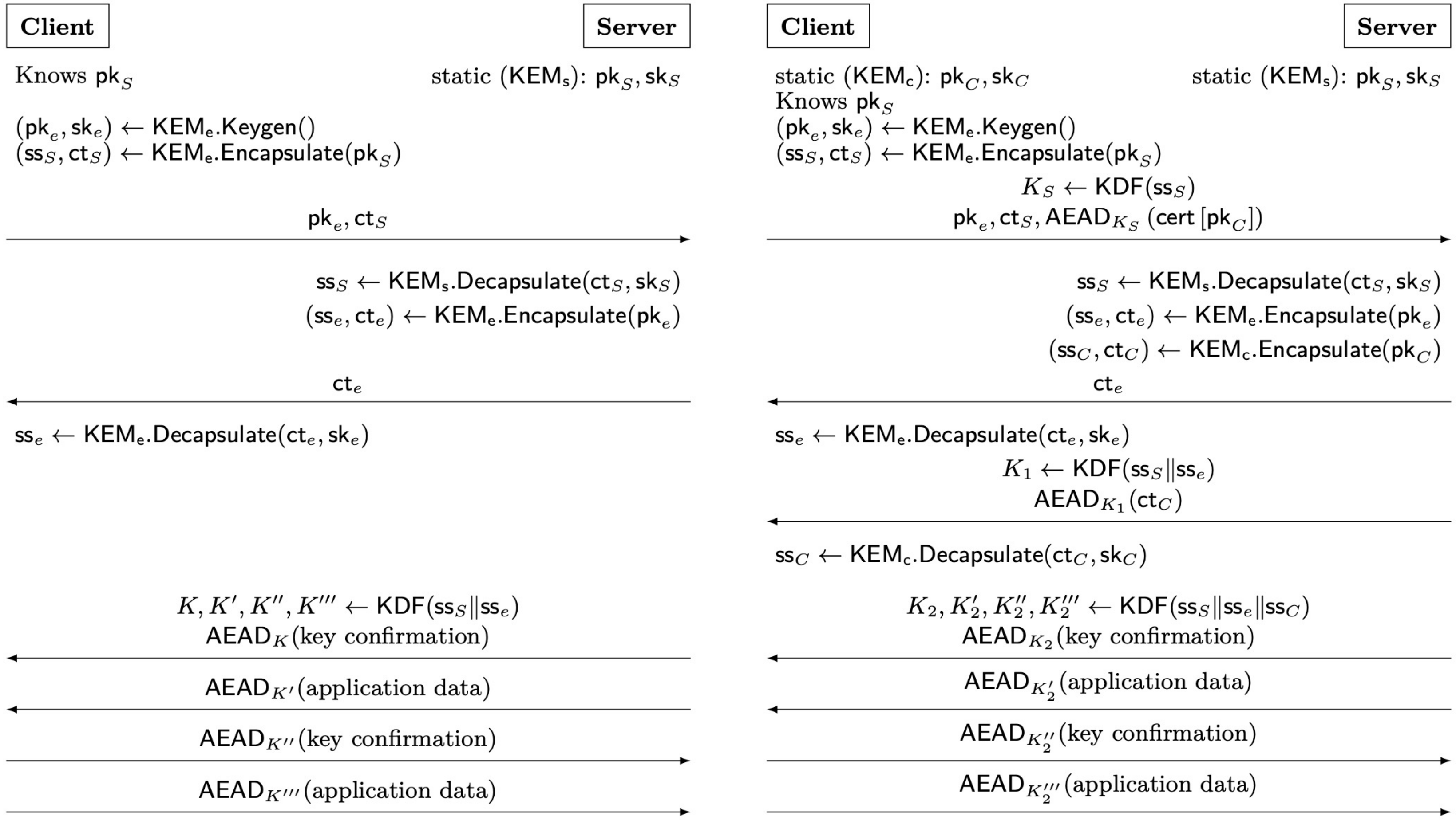
TLS 1.3 and KEMTLS crypto & handshake time

		Computation time for asymmetric crypto				Handshake time (31.1 ms latency, 1000 Mbps bandwidth)						Handshake time (195.6 ms latency, 10 Mbps bandwidth)					
		Excl. int. CA cert.		Incl. int. CA cert.		Excl. int. CA cert.			Incl. int. CA cert.			Excl. int. CA cert.			Incl. int. CA cert.		
		Client	Server	Client	Server	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done	Client sent req.	Client recv. resp.	Server HS done
TLS 1.3	**ERRR**	0.134	0.629	0.150	0.629	66.4	**97.6**	35.4	66.6	**97.8**	35.6	397.1	**593.3**	201.3	398.2	**594.3**	202.3
	SFXG	40.058	21.676	40.094	21.676	165.8	**196.9**	134.0	166.2	**197.3**	134.4	482.1	**678.4**	285.8	482.5	**678.8**	286.2
	SFGG	34.104	21.676	34.141	21.676	154.9	**186.0**	123.1	259.0	**290.2**	227.1	473.7	**669.8**	277.5	10936.3	**11902.5**	10384.1
	KDDD	0.080	0.087	0.111	0.087	64.3	**95.5**	33.3	64.8	**96.0**	33.8	411.6	**852.4**	446.1	415.9	**854.7**	448.0
	NFFF	0.141	0.254	0.181	0.254	65.1	**96.3**	34.1	65.6	**96.9**	34.7	398.1	**662.2**	269.2	406.7	**842.8**	443.5
KEMTLS	**SSXG**	61.456	41.712	61.493	41.712	202.1	**268.8**	205.6	202.3	**269.1**	205.9	505.8	**732.0**	339.7	506.1	**732.4**	340.1
	SSGG	55.503	41.712	55.540	41.712	190.4	**256.6**	193.4	293.3	**359.5**	296.3	496.8	**723.0**	330.8	10859.5	**11861.0**	10331.7
	KKDD	0.060	0.021	0.091	0.021	63.4	**95.0**	32.7	63.9	**95.5**	33.2	399.2	**835.1**	439.9	418.9	**864.2**	447.6
	NNFF	0.118	0.027	0.158	0.027	63.6	**95.2**	32.9	64.2	**95.8**	33.5	396.2	**593.4**	200.6	400.0	**835.6**	440.2

Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate CA certificate, D = root certificate.

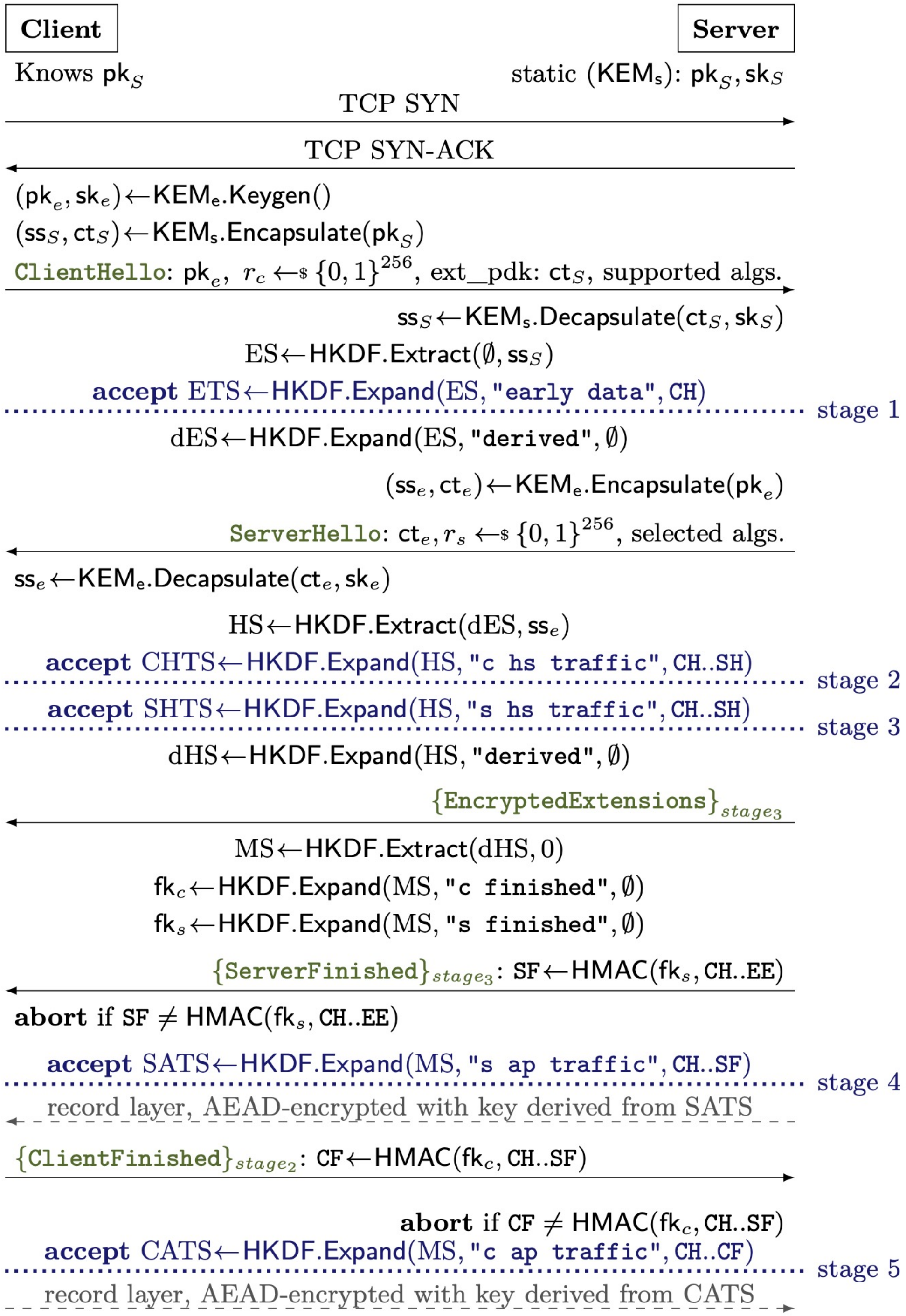
Label values: Dilithium, ECDH X25519, Falcon, GeMSS, Kyber, NTRU, RSA-2048, SIKE, $XMSS_s^{MT}$; all level-1 schemes.

KEMTLS-PDK overview

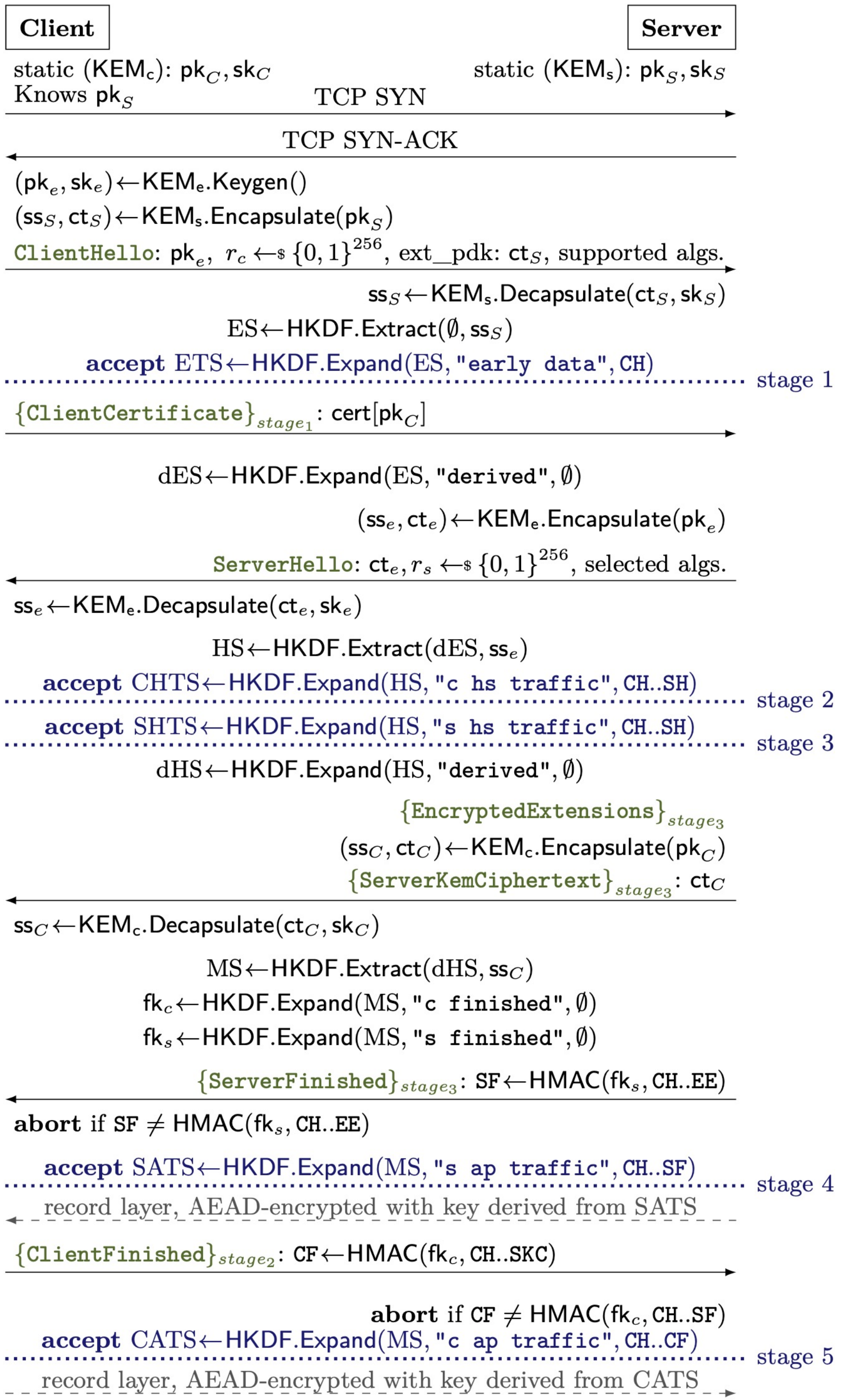


(a) Unilaterally authenticated (b) With proactive client authentication

KEMTLS-PDK


Client
Server
Knows pk_S
static (KEM_s): pk_S, sk_S
TCP SYN
TCP SYN-ACK
(pk_e, sk_e) ← KEM_e.Keygen()
(ss_S, ct_S) ← KEM_s.Encapsulate(pk_S)
ClientHello: pk_e, r_c ←$ {0,1}^256, ext_pdk: ct_S, supported algs.
ss_S ← KEM_s.Decapsulate(ct_S, sk_S)
ES ← HKDF.Extract(∅, ss_S)
accept ETS ← HKDF.Expand(ES, "early data", CH)
stage 1
dES ← HKDF.Expand(ES, "derived", ∅)
(ss_e, ct_e) ← KEM_e.Encapsulate(pk_e)
ServerHello: ct_e, r_s ←$ {0,1}^256, selected algs.
ss_e ← KEM_e.Decapsulate(ct_e, sk_e)
HS ← HKDF.Extract(dES, ss_e)
accept CHTS ← HKDF.Expand(HS, "c hs traffic", CH..SH)
stage 2
accept SHTS ← HKDF.Expand(HS, "s hs traffic", CH..SH)
stage 3
dHS ← HKDF.Expand(HS, "derived", ∅)
{EncryptedExtensions}_stage3
MS ← HKDF.Extract(dHS, 0)
fk_c ← HKDF.Expand(MS, "c finished", ∅)
fk_s ← HKDF.Expand(MS, "s finished", ∅)
{ServerFinished}_stage3: SF ← HMAC(fk_s, CH..EE)
abort if SF ≠ HMAC(fk_s, CH..EE)
accept SATS ← HKDF.Expand(MS, "s ap traffic", CH..SF)
stage 4
record layer, AEAD-encrypted with key derived from SATS
{ClientFinished}_stage2: CF ← HMAC(fk_c, CH..SF)
abort if CF ≠ HMAC(fk_c, CH..SF)
accept CATS ← HKDF.Expand(MS, "c ap traffic", CH..CF)
stage 5
record layer, AEAD-encrypted with key derived from CATS

KEMTLS-PDK with proactive client authentication



Communication sizes

KEMTLS

TLS 1.3 w/cached server certs

KEMTLS-PDK

		Transmitted Ephem. (pk+ct)	Transmitted Auth	Sum	Client Auth Cert. (pk+ct/sig)	Client Auth CA (sig)	Sum (total)	Cached Leaf pk	Cached Cl. Auth CA (pk)
KEMTLS	Minimum	SIKE 197 236	SIKE/Rai. crt+ct 499	**932**	SIKE 433	Rainbow 66	**1,431**	N/A	Rainbow 161,600
KEMTLS	Assumption: MLWE/MSIS	Kyber 800 768	Kyber/Dil. crt+ct 3,988	**5,556**	Kyber 1,568	Dilithium 2,420	**9,554**	N/A	Dilithium 1,312
KEMTLS	Assumption: NTRU	NTRU 699 699	NTRU/Fal. crt+ct 2,088	**3,486**	NTRU 1,398	Falcon 690	**5,574**	N/A	Falcon 897
Cached TLS	TLS 1.3	X25519 32 32	RSA-2048 sig 256	**320**	RSA-2048 528	RSA-2048 256	**1,104**	RSA-2048 272	RSA-2048 272
Cached TLS	Minimum	SIKE 197 236	Rainbow sig 66	**499**	Falcon 1,587	Rainbow 66	**2,152**	Rainbow 161,600	Rainbow 161,600
Cached TLS	Assumption: MLWE/MSIS	Kyber 800 768	Dilithium sig 2,420	**3,988**	Dilithium 3,732	Dilithium 2,420	**10,140**	Dilithium 1,312	Dilithium 1,312
Cached TLS	Assumption: NTRU	NTRU 699 699	Falcon sig 690	**2,088**	Falcon 1,587	Falcon 690	**4,365**	Falcon 897	Falcon 897
KEMTLS-PDK	Minimum	SIKE 197 236	McEliece ct 128	**561**	SIKE 433	Rainbow 66	**1,060**	McEliece 261,120	Rainbow 161,600
KEMTLS-PDK	Finalist: Kyber	Kyber 800 768	Kyber ct 768	**2,336**	Kyber 1,568	Dilithium 2,420	**6,324**	Kyber 800	Dilithium 1,312
KEMTLS-PDK	Finalist: NTRU	NTRU 699 699	NTRU ct 699	**2,097**	NTRU 1,398	Falcon 690	**4,185**	NTRU 699	Falcon 897
KEMTLS-PDK	Finalist: SABER	SABER 672 736	SABER ct 736	**2,144**	SABER 1,408	Dilithium 2,420	**5,972**	SABER 672	Dilithium 1,312

Handshake times, unilateral authentication

	Unilaterally authenticated	31.1 ms RTT, 1000 Mbps Client sent req.	Client recv. resp.	Server expl. auth.	195.6 ms RTT, 10 Mbps Client sent req.	Client recv. resp.	Server expl. auth.
KEMTLS	Minimum	75.4	**116.1**	116.1	408.6	**616.3**	616.2
	MLWE/MSIS	63.2	**94.8**	94.7	397.4	**594.6**	594.5
	NTRU	63.1	**94.7**	94.6	396.0	**593.0**	593.0
Cached TLS	TLS 1.3	66.4	**97.6**	66.3	396.8	**592.9**	396.7
	Minimum	70.1	**101.3**	70.0	402.3	**598.5**	402.2
	MLWE/MSIS	63.9	**95.1**	63.8	397.2	**593.4**	397.1
	NTRU	64.8	**96.1**	64.7	397.0	**593.2**	396.9
PDK	Minimum	66.3	**97.5**	66.2	397.9	**594.1**	397.8
	Kyber	63.1	**94.3**	63.0	395.3	**591.4**	395.2
	NTRU	63.1	**94.3**	63.0	395.3	**591.5**	395.2
	SABER	63.1	**94.3**	63.0	395.2	**591.4**	395.2

Handshake times, mutual authentication

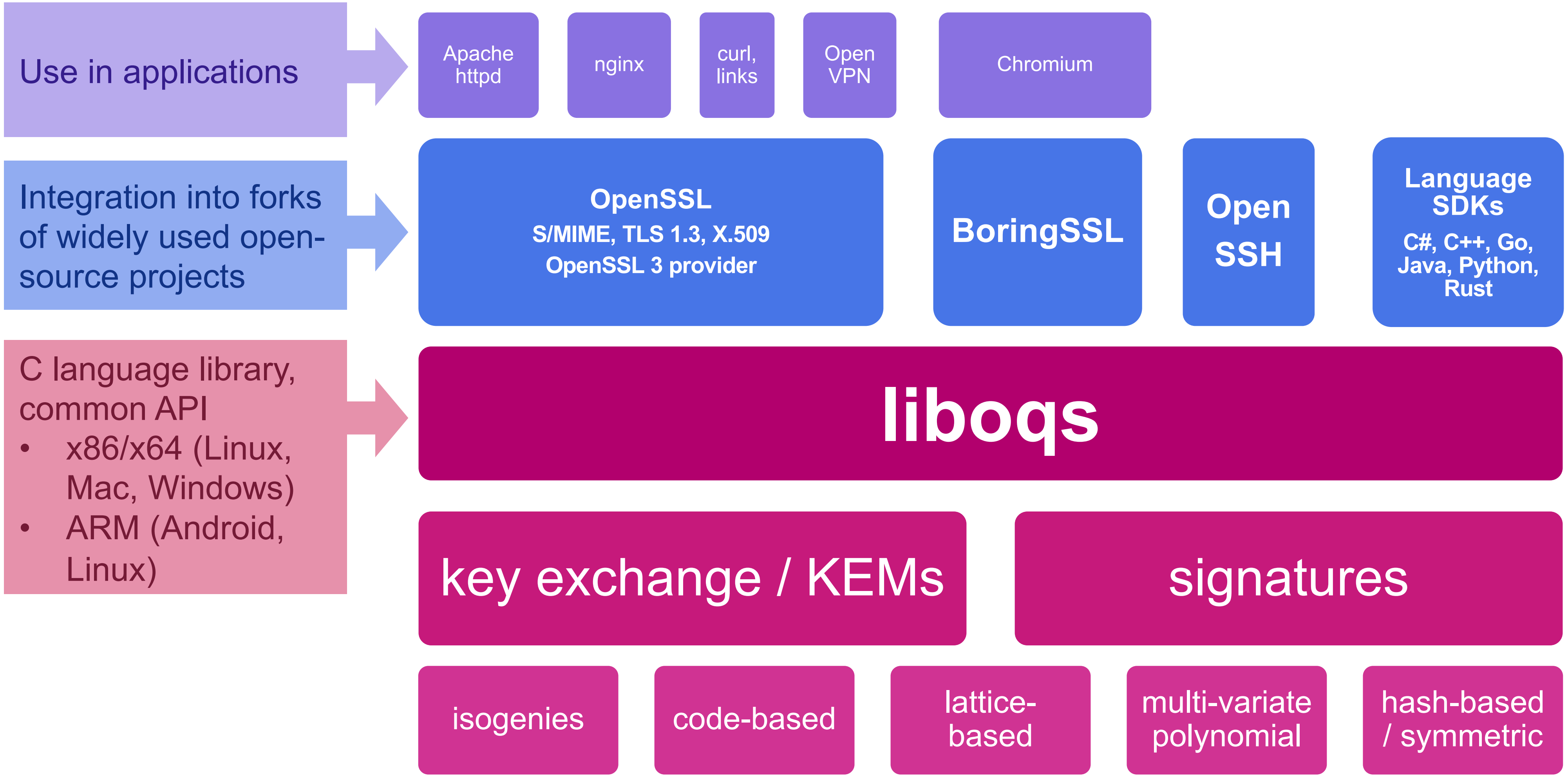
	Mutually authenticated	31.1 ms RTT, 1000 Mbps Client sent req.	Client recv. resp.	Server expl. auth.	195.6 ms RTT, 10 Mbps Client sent req.	Client recv. resp.	Server expl. auth.
KEMTLS	Minimum	130.2	**161.4**	161.3	631.2	**827.5**	827.5
KEMTLS	MLWE/MSIS	95.2	**126.6**	126.6	598.3	**794.6**	794.6
KEMTLS	NTRU	95.0	**126.4**	126.3	595.3	**791.7**	791.7
Cached TLS	TLS 1.3	68.3	**99.8**	65.9	399.4	**597.2**	396.7
Cached TLS	Minimum	71.1	**102.7**	69.9	403.3	**602.0**	402.0
Cached TLS	MLWE/MSIS	64.5	**96.2**	63.9	400.1	**616.8**	399.5
Cached TLS	NTRU	66.2	**98.1**	64.8	398.3	**597.7**	397.0
PDK	Minimum	84.9	**116.1**	84.9	420.5	**616.8**	420.5
PDK	Kyber	63.5	**94.7**	63.4	400.2	**596.5**	400.2
PDK	NTRU	63.6	**94.9**	63.6	397.6	**593.8**	397.5
PDK	SABER	63.6	**94.8**	63.5	399.4	**595.5**	399.3

OPEN QUANTUM SAFE

software for prototyping quantum-resistant cryptography

https://openquantumsafe.org https://github.com/open-quantum-safe

Open Quantum Safe Project



Industry partners:
- Amazon Web Services
- evolutionQ
- IBM Research
- Microsoft Research

Additional contributors:
- Cisco
- Senetas
- PQClean project
- Individuals

Financial support:
- AWS
- Canadian Centre for Cyber Security
- NSERC
- Unitary Fund

https://openquantumsafe.org/ • https://github.com/open-quantum-safe/

liboqs

- C library with common API for post-quantum signature schemes and key encapsulation mechanisms
- MIT License
- Builds on Windows, macOS, Linux; x86_64, ARM v8

- Version 0.5.0 released March 2021
- Includes all Round 3 finalists and alternate candidates
  - (except GeMSS)
  - Some implementations still Round 2 versions

https://openquantumsafe.org/liboqs/

TLS 1.3 implementations

	OQS-OpenSSL 1.1.1	OQS-OpenSSL 3 provider	OQS-BoringSSL
PQ key exchange in TLS 1.3	Yes	Yes	Yes
Hybrid key exchange in TLS 1.3	Yes	Coming soon	Yes
PQ certificates and signature authentication in TLS 1.3	Yes	No	Yes
Hybrid certificates and signature authentication in TLS 1.3	Yes	No	No

Using draft-ietf-tls-hybrid-design for hybrid key exchange

Interoperability test server running at https://test.openquantumsafe.org

https://openquantumsafe.org/applications/tls/

Applications

- Demonstrator application integrations into:
  - Apache
  - nginx
  - haproxy
  - curl
  - Chromium
- In most cases required few/no modifications to work with updated OpenSSL
- Runnable Docker images available for download

https://openquantumsafe.org/applications/tls/#demo-integrations

Benchmarking

- New benchmarking portal at https://openquantumsafe.org/benchmarking/

- Core algorithm speed and memory usage
- TLS performance in ideal network conditions
- Intel AVX2 and ARM 64