Recent results for KEMTLS

Douglas Stebila

University of Waterloo

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
- Elliptic curves: David Jao, Alfred Menezes, (Scott Vanstone)
- More cryptography: Sergey Gorbunov, Mohammad Hajiabadi, Doug Stinson
- Privacy-enhancing technologies: Ian Goldberg
- Quantum cryptanalysis: Michele Mosca
- Quantum cryptography: Norbert Lütkenhaus, Thomas Jennewein, Debbie Leung
- Even more cryptography and security: Gord Agnew, Vijay Ganesh, Guang Gong, Sergey Gorbunov, Anwar Hasan, Florian Kerschbaum

KEMTLS

Reimagining of TLS 1.3 handshake to use key encapsulation mechanisms (KEMs) for implicit authentication, rather than digital signatures for explicit authentication

- Reduce communication sizes in PQ setting since PQ KEMs are in general smaller than PQ signatures
- Can reduce computation costs in some configurations

Outline

1. KEMTLS design and performance
2. Pre-distributed public keys for faster client authentication
3. Proving KEMTLS manually and with Tamarin
4. Certificate lifecycle for KEM public keys

1. KEMTLS design and performance

Peter Schwabe, Douglas Stebila, Thom Wiggers
ACM CCS 2020. https://eprint.iacr.org/2020/534

Sofía Celi, Peter Schwabe, Douglas Stebila, Nick Sullivan, Thom Wiggers.
https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00

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	$\sigma_B \leftarrow \text{SIG.Sign}(sk_B, A \Vert B \Vert X \Vert Y)$
$\sigma_A \leftarrow \text{SIG.Sign}(sk_A, A \Vert B \Vert X \Vert Y)$	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

$sk_A \leftarrow_\$ \{0, \ldots, q-1\}$

$pk_A \leftarrow g^{sk_A}$

obtain pk_B

$x \leftarrow_\$ \{0, \ldots, q-1\}$

$X \leftarrow g^x$

Bob

$sk_B \leftarrow_\$ \{0, \ldots, q-1\}$

$pk_B \leftarrow g^{sk_B}$

obtain pk_A

Static DH

$y \leftarrow_\$ \{0, \ldots, q-1\}$

$Y \leftarrow g^y$

Ephemeral DH

Alice → Bob: X

Bob → Alice: Y

Alice: $k \leftarrow H(sid, pk_B^{sk_A} \| Y^x)$

Bob: $k \leftarrow H(sid, pk_A^{sk_B} \| X^y)$

Alice ⇔ Bob: application data, using authenticated encryption

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**
Rainbow	**Multi-variate**	**60,192**	**66**

Solution

use
post-quantum KEMs
for authentication

Key encapsulation mechanisms (KEMs)

An abstraction of Diffie–Hellman key exchange

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

$\xrightarrow{\quad pk \quad}$

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

$\xleftarrow{\quad ct \quad}$

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

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**
Rainbow	**Multi-variate**	**60,192**	**66**

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

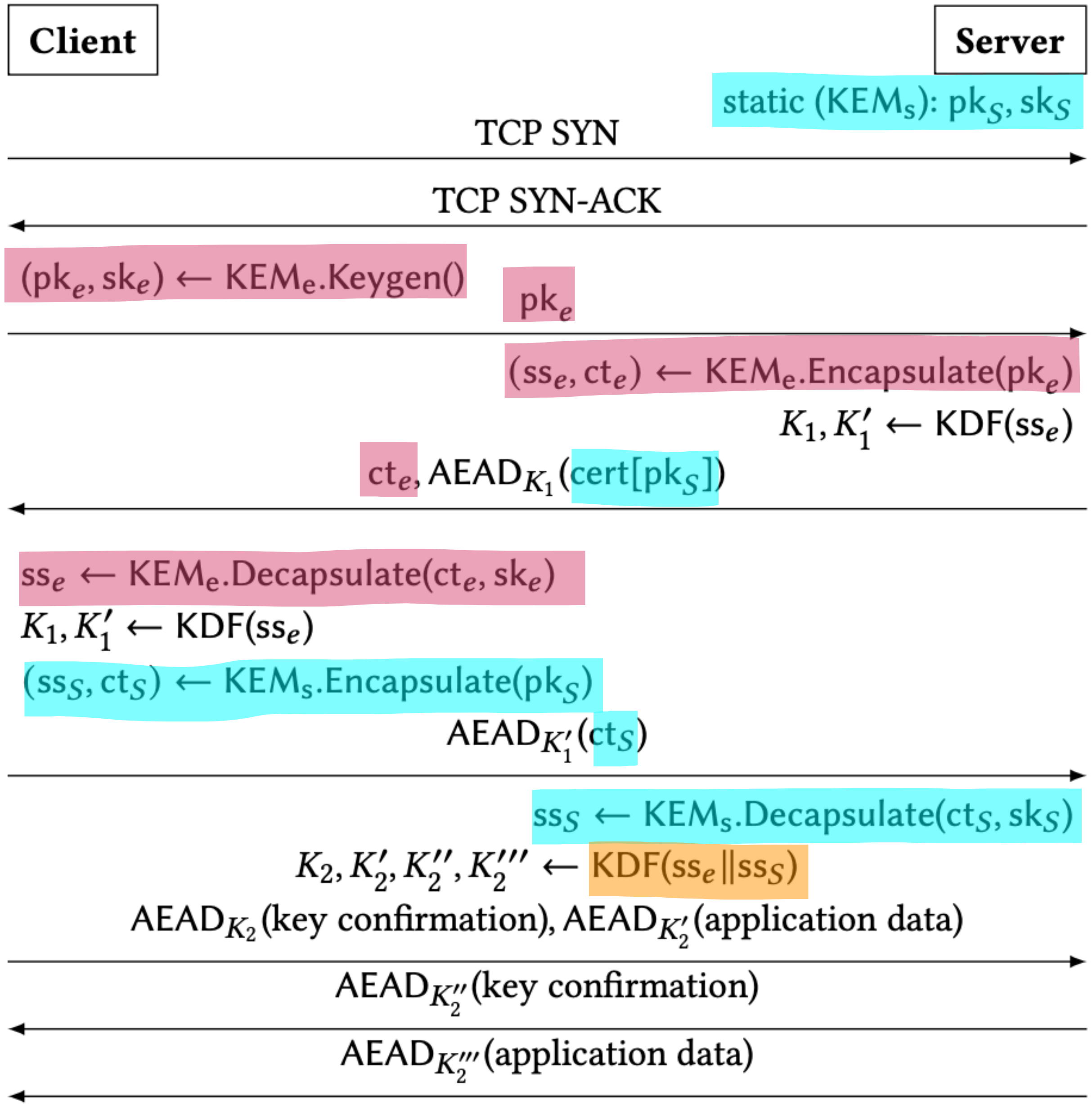
In theory

- 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

KEM for authenticated key exchange

Signature scheme for intermediate CA

Signature scheme for root CA

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

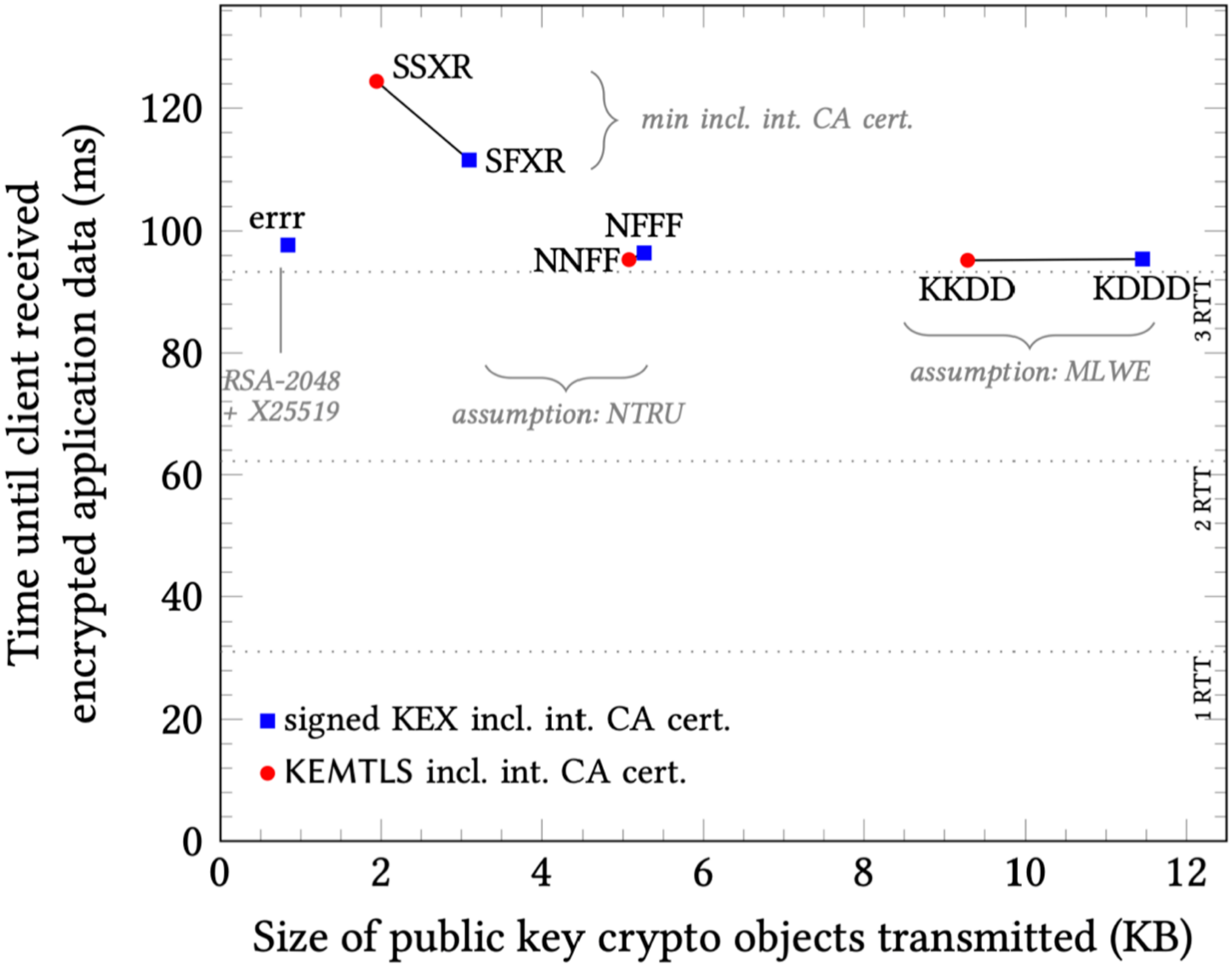
4 scenarios

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,
Kyber,
NTRU,
Rainbow,
rSA-2048,
SIKE,
XMSS'

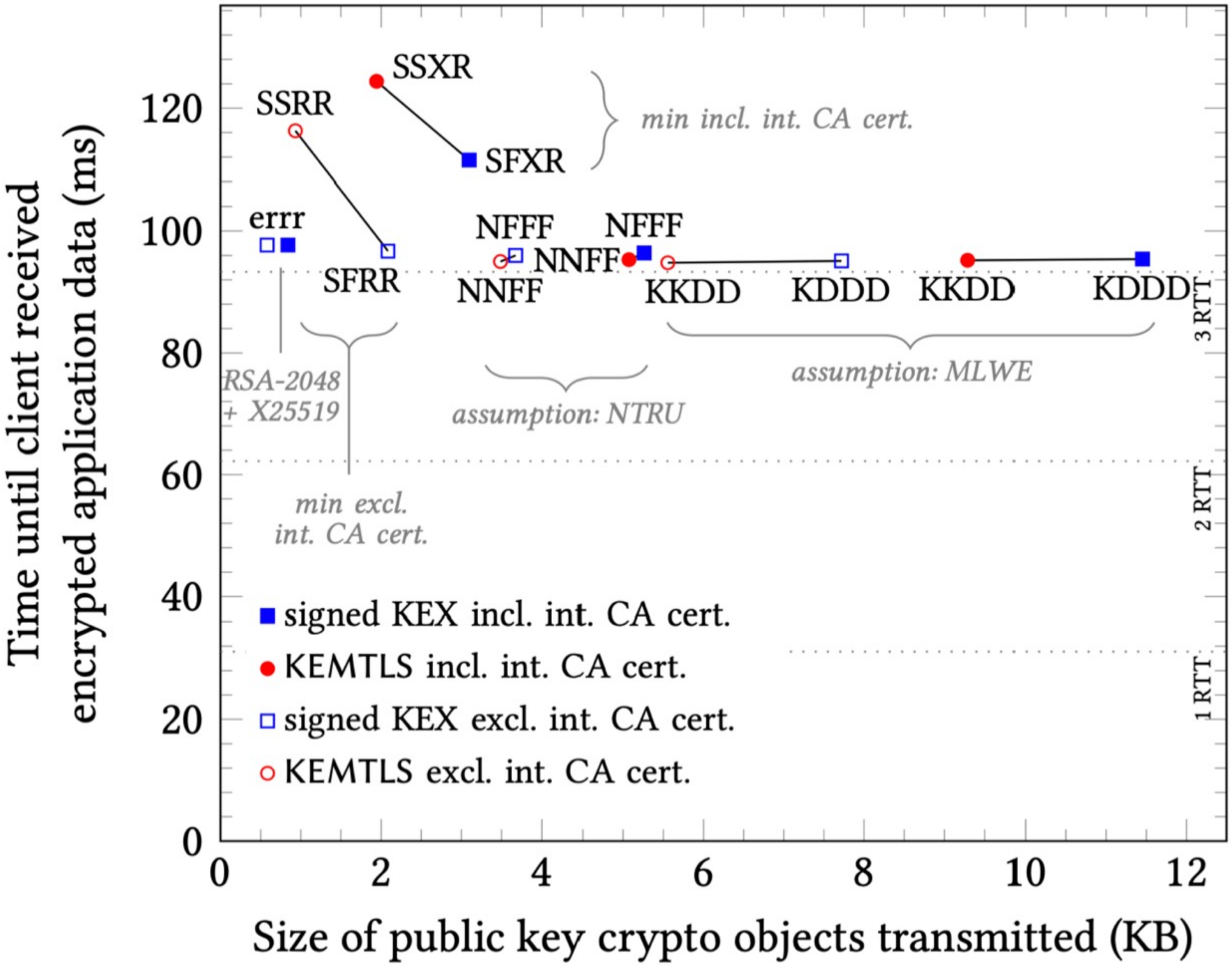


Rustls client/server with some AVX2 implementations. Emulated network: latency 31.1 ms, bandwidth 1000 Mbps, 0% packet loss. Average of 100000 iterations.

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,
Kyber,
NTRU,
Rainbow,
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 in server-only auth mode
- Smaller trusted code base (no signature generation on client/server)

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

- Preserves client confidentiality
- Adds extra round trip

2. Pre-distributed public keys for faster client authentication

Peter Schwabe, Douglas Stebila, Thom Wiggers
ESORICS 2021. https://eprint.iacr.org/2021/779

Sofía Celi, Peter Schwabe, Douglas Stebila, Nick Sullivan, Thom Wiggers.
https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00

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

Different from TLS 1.3 pre-shared symmetric key mode

- PSK is a harder(?) key management problem
- Different compromise model

Variant: Pre-distributed public keys

- Alternate KEMTLS protocol flow when server certificates are known in advance
- Resumption-style mechanism that avoids the downsides of symmetric-key TLS PSK
- Given server's long-term key, client can send ciphertext in ClientHello
- Also allow to send client certificate in ClientHello

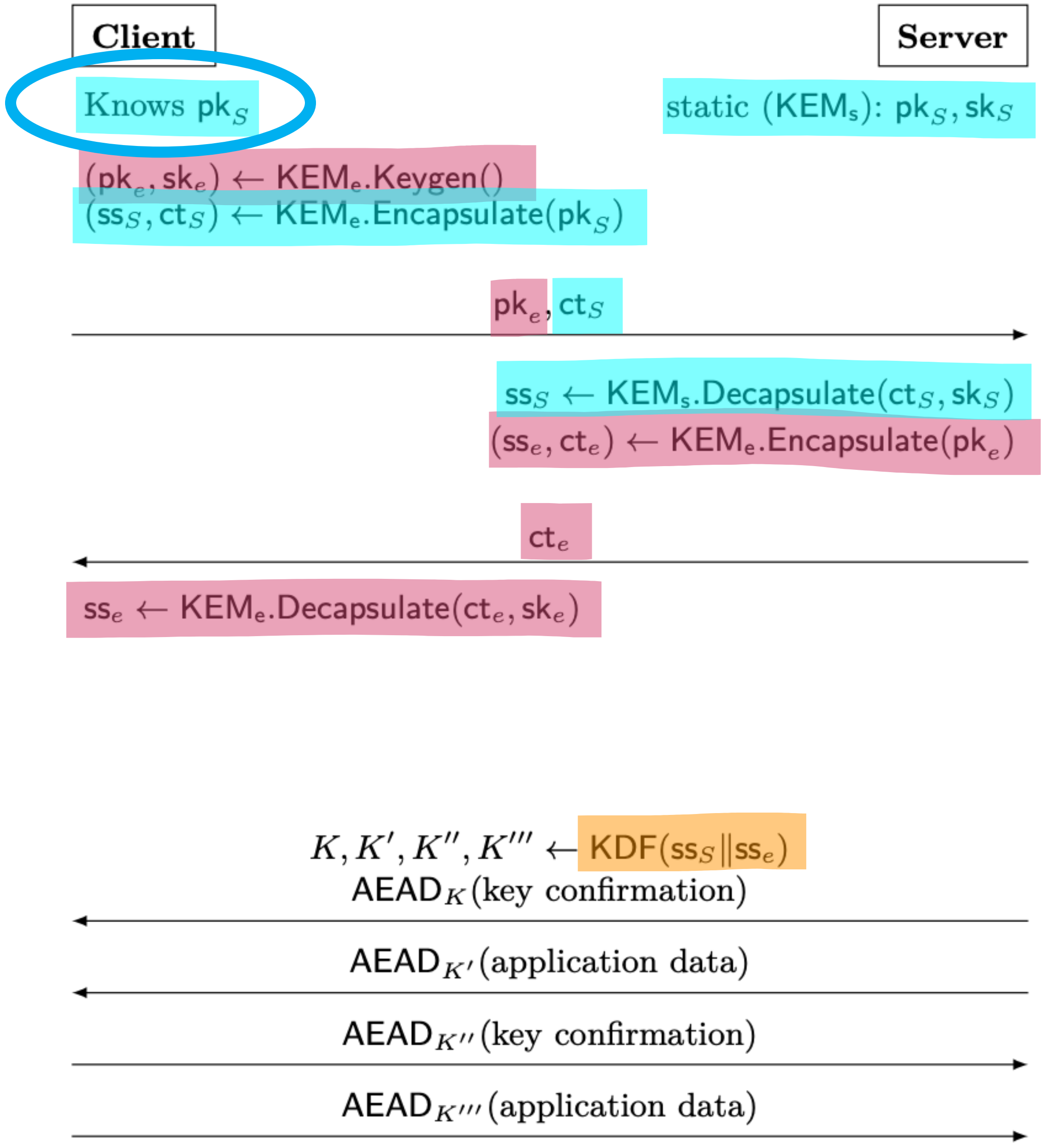
Get a **1-RTT**,
TLS 1.3-shape
handshake with
implicit authentication

KEMTLS-PDK handshake server-only auth.

KEM for ephemeral key exchange

KEM for server-to-client authenticated key exchange

Combine shared secrets



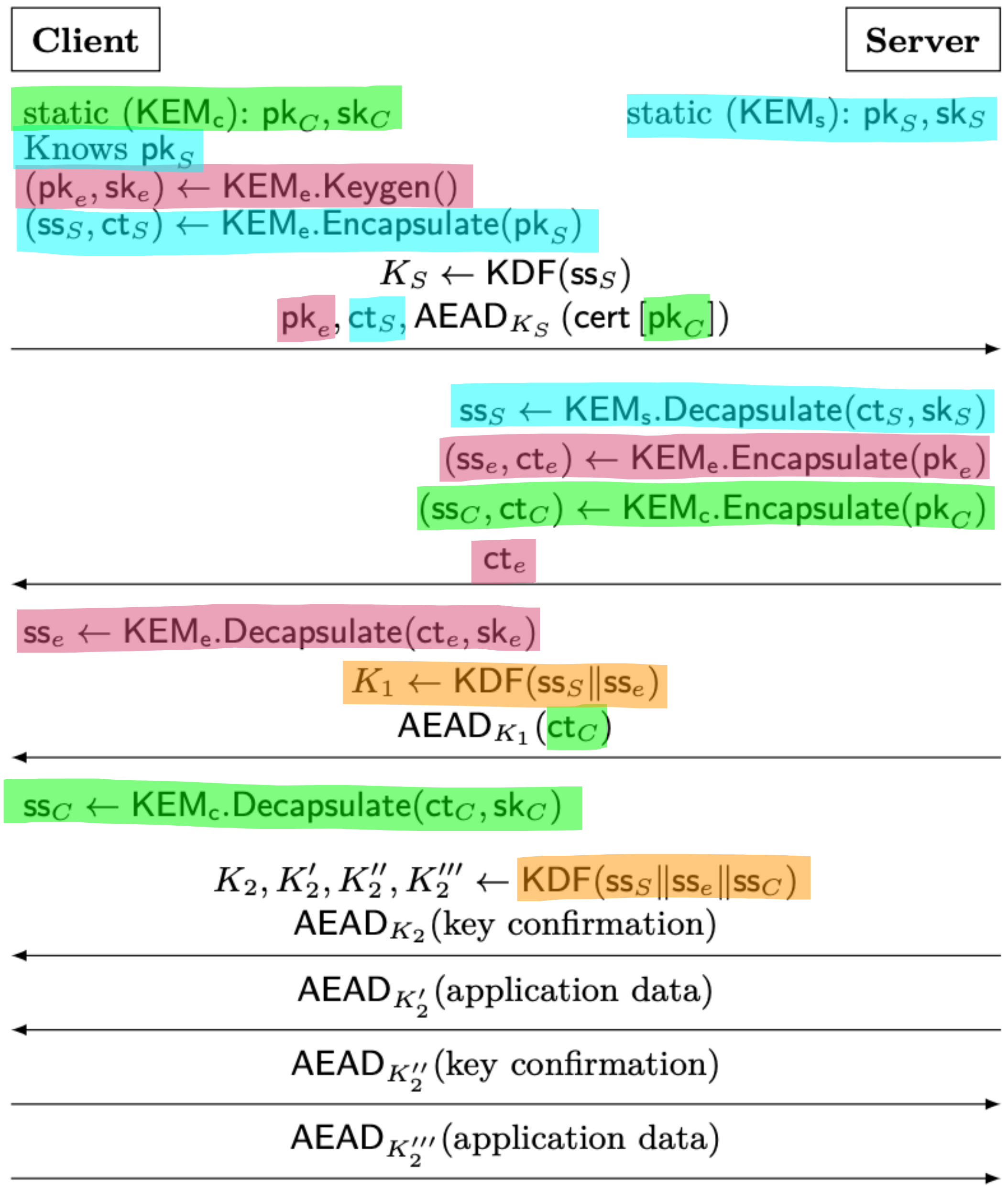
KEMTLS-PDK handshake mutual auth

KEM for ephemeral key exchange

KEM for server-to-client authenticated key exchange

KEM for client-to-server authenticated key exchange

Combine shared secrets



Benefits from pre-distributed key variant

- 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 variants

Traditional communication flow:

1. KEMTLS server-only authentication
2. KEMTLS mutual authentication

Pre-distributed server public keys:

3. KEMTLS-PDK server-only authentication
4. KEMTLS-PDK mutual authentication

KEMTLS: ACM CCS 2020 https://eprint.iacr.org/2020/534
KEMTLS-PDK: ESORICS 2021 https://eprint.iacr.org/2021/779

3. Proving KEMTLS manually and with Tamarin

Peter Schwabe, Douglas Stebila, Thom Wiggers
ACM CCS 2020. https://eprint.iacr.org/2020/534

Peter Schwabe, Douglas Stebila, Thom Wiggers
ESORICS 2021. https://eprint.iacr.org/2021/779

Sofía Celi, Jonathan Hoyland, Douglas Stebila, Thom Wiggers
Coming soon to an eprint server near you!
https://github.com/thomwiggers/TLS13Tamarin
https://github.com/dstebila/KEMTLS-Tamarin/

Security properties

Key indistinguishability

Forward secrecy

Implicit and explicit authentication

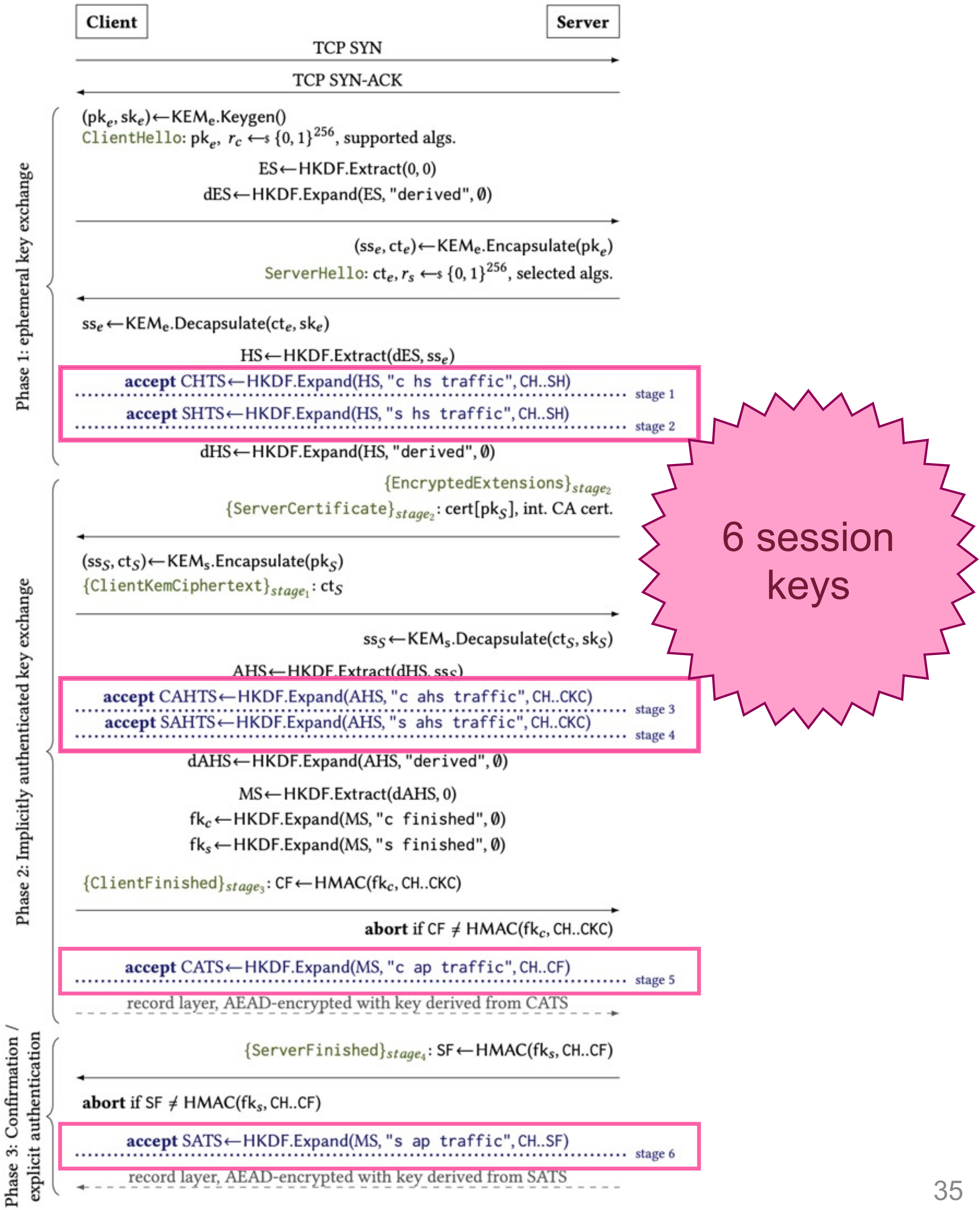
Deniability

Multi-stage authenticated key exchange model for KEMTLS

→ Bellare–Rogaway AKE model

→ Multi-stage AKE model [FG14]

→ Multi-stage AKE model for TLS 1.3 [DFGS15]



[BR93] Bellare, Rogaway, Crypto'93. [FG14] Fischlin, Günther, ACM CCS 2014.
[DFGS15] Dowling, Fischlin, Günther, Stebila, ACM CCS 2015.

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

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

Security subtleties: 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 judge

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

Security analyses of KEMTLS & KEMTLS-PDK

Pen-and-paper

- Proves session key security and authentication in the multi-stage key exchange model
- Using provable security paradigm

Formal verification

Using Tamarin prover
(a symbolic model checker):

1. Adaptation of full-scale TLS 1.3 Tamarin model of [CHHSV] to capture KEMTLS & KEMTLS-PDK
2. Tamarin analog of pen-and-paper multi-stage key exchange model

Tamarin prover: https://tamarin-prover.github.io/
[CHHSV] Cremers, Horvat, Hoyland, Scott, van der Merwe. ACM CCS 2017.

Pen and paper proof in the multi-stage model

Session key indistinguishability

- For every stage key
- With 1 of 3 levels of forward secrecy varying by stage
- Retroactive upgrade
- Adversary powers:
  - Network control
  - Corrupt long-term keys
  - Reveal session keys

Authentication ("malicious acceptance")

- Expectations varying by stage
- Retroactive upgrade
- Includes replayability (non-uniqueness) for some PDK stages

Cryptographic assumptions

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

[BR93] Bellare, Rogaway, Crypto'93. [FG14] Fischlin, Günther, ACM CCS 2014.
[DFGS15] Dowling, Fischlin, Günther, Stebila, ACM CCS 2015.

Limitations of pen-and-paper proofs

- Fully written out for session-key indistinguishability for KEMTLS and KEMTLS-PDK server-only auth variants
  - But only as reliable as the authors and the readers are
- Proof sketches for session-key indistinguishability of remaining variants
- Hand-waving argument for offline deniability

Formal verification using Tamarin

- Tamarin prover is a model checker for security protocols in the symbolic model
- Protocol and adversary powers are specified as a set of state machine transitions (“multiset rewriting rules”)
- Security property is specified as a predicate over actions recorded during state machine transitions
- Tamarin prover explores (infinite) state space of all possible executions to find an execution trace that violates the security property or verifies that none exists (or fails to terminate)

https://tamarin-prover.github.io/

Formal verification using Tamarin

- Tamarin successfully used on many academic and real-world cryptographic protocols
- Especially effective on key exchange protocols
  - Note Tamarin models key exchange security based on *learning* session key, not *indistinguishability*

- Tamarin model of TLS 1.3 drafts [CHSV,CHHSV] found several flaws
  - Especially in interactions between different protocols modes
    - e.g. in TLS 1.3 pre-shared key resumption
  - Expensive: months of person-effort, 1 week of computation time, 100 GB RAM

[CHSV] Cremers, Horvat, Scott, van der Merwe, IEEE S&P 2016.
[CHHSV] Cremers, Horvat, Hoyland, Scott, van der Merwe, ACM CCS 2017.

Modelling KEMTLS using Tamarin

Approach 1

https://github.com/thomwiggers/TLS13Tamarin

- Adapt [CHHSV] full-scale Tamarin model of TLS 1.3 to KEMTLS
- High resolution protocol specification: captures TLS message format, internal KDF structure, …
- Lower resolution security properties
- Required more human effort to get proofs running automatically

Approach 2

https://github.com/dstebila/KEMTLS-Tamarin

- Encode pen-and-paper multi-stage AKE definitions in Tamarin
- Lower resolution protocol specification: "core cryptographic" of KEMTLS
  - E.g. No TLS message structure
- Higher resolution security properties
- Simpler to specify and automatically proves

[CHHSV] Cremers, Horvat, Hoyland, Scott, van der Merwe, ACM CCS 2017.

Feature	**In model of** **Approach 1**	**Approach 2**
Protocol modelling		
Encrypted handshake messages	✓	✗
HKDF and HMAC decomposed into hash calls	✓	✗
Key exch. and auth. KEMs are the same algorithm	✓	✗
TLS message structure	✓	✗
Security properties		
Adversary can reveal long-term keys	✓	✓
Adversary can reveal ephemeral keys	✓	✗
Adversary can reveal intermediate session keys	✗	✓
Secrecy of handshake and application traffic keys	✓	✓
Forward secrecy	✓	✓
Multiple flavours of forward secrecy	✗	✓
Explicit authentication	✓	✓
Deniability	✗	✓

Lessons learned from formal verification

- Higher assurance in protocol design
- Captures potential interactions between all 4 protocol variants
- Exhibits difficulty trade-off in formal verification: granularity of protocol specification
  versus
  granularity of security properties
- Formal verification identified bugs in previous work:
  - Approach 1 identified minor bugs in original TLS 1.3 Tamarin model of [CHHSV]
  - Approach 2 identified minor bugs in security properties stated in original KEMTLS and KEMTLS-PDK papers
    - E.g. Wrong retroactive authentication stages or incorrect forward secrecy levels for some stages

4. Certificate lifecycle for KEM public keys

Tim Güneysu, Philip Hodges, Georg Land, Mike Ounsworth, Douglas Stebila, Greg Zaverucha
Coming soon to an eprint server near you!

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
- …

Certificate lifecycle

Certificate requests in the X.509 PKI

How does requester prove possession of corresponding secret keys?

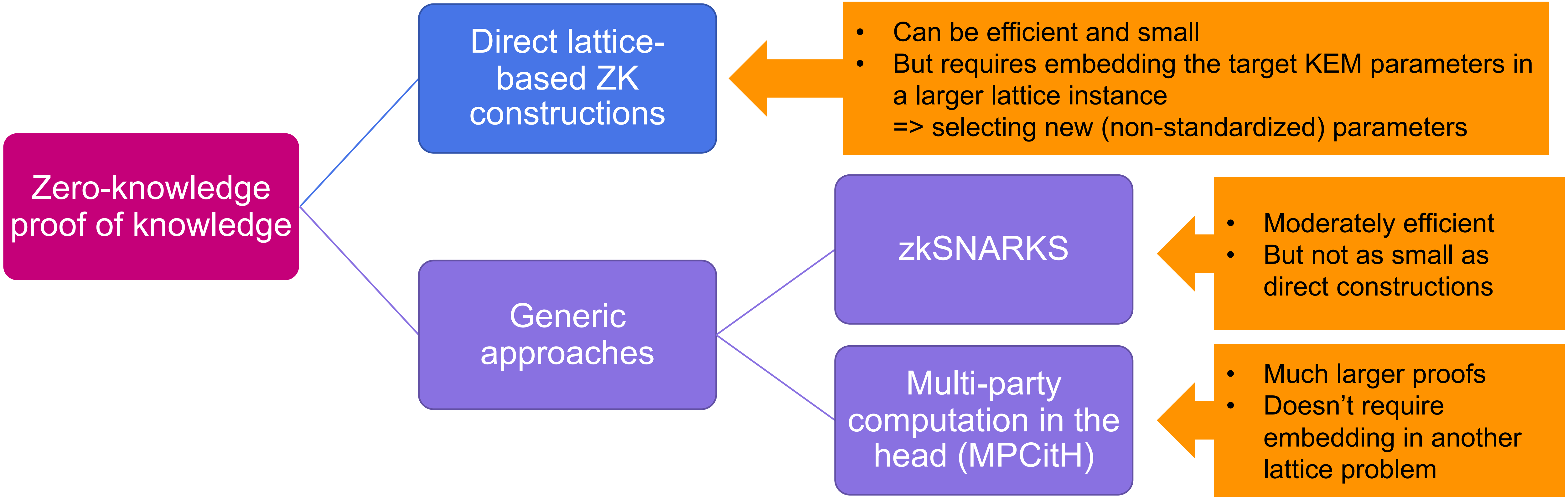
1. Interactive challenge-response protocol [RFC 4210 Sect. 5.2.8.3]
2. Send certificate back encrypted under subject public key [RFC 4210 Sect. 5.2.8.2]
   - Weird confidentiality requirement on certificate.
   - Maybe broken by Certificate Transparency or other logging mechanisms?
3. Non-interactive certificate signing requests [RFC 2986]
   - CSRs okay for signature schemes, but not for public key encryption or key encapsulation mechanisms

Goal:
Design non-interactive proof of possession for lattice-based KEM public keys

(so that we can have the same certificate lifecycle
for KEM certificates to enable KEMTLS)

lattice-based = FrodoKEM (plain LWE), Kyber (module LWE)

Possible approaches for non-interactive proof of possession for (lattice-based) KEM public keys



Our approach

Proof of possession via ***verifiable generation***

Generate the key and a proof at the same time

FrodoKEM key generation

1. Generate $\mathbf{A} \in \mathbb{Z}_q^{n \times n}$ from a seed
2. Sample $\mathbf{S} \leftarrow_\$ \chi^{n \times \overline{n}}$
3. Sample $\mathbf{E} \leftarrow_\$ \chi^{n \times \overline{n}}$
4. Compute $\mathbf{B} \leftarrow_\$ \mathbf{AS} + \mathbf{E}$
5. Public key: $(\mathsf{seed}_{\mathbf{A}}, \mathbf{B})$
6. Secret key: $\mathbf{S}$

<u>Frodo-640</u>

$q = 2^{15}$

$n = 640, \overline{n} = 8$

$\chi \in [-12, \ldots, 12]$

https://frodokem.org/

Verifiable generation for FrodoKEM

1. Generate and commit to many allegedly small values for S and E
2. Reveal some of them to prove they're small
3. Use the rest for the actual key generation

This doesn't prove that all the unrevealed values are small, only most of them with high probability

How do we prove we actually used them in the rest of the key generation?

- MPC-in-the-head à la Picnic
- Fiat–Shamir to get a signature scheme

5-round interactive protocol for verifiable generation

1. Prover: Generate sufficiently many small values.
   Generate an additive secret sharing among N parties.
   Commit to the shares.
   Send commitments.
2. Verifier: Pick some fraction of the bundles to audit.
3. Prover: Open commitments for challenged bundles.
   Use unaudited bundles to construct secret key (S, E) and public key B=AS+E. Commit to shares of B.
   Send commitments and public key (A, B).
4. Verifier: Select N-1 parties to audit.
5. Prover: Reveal state of N-1 parties.
6. Verifier: Check state of revealed parties.

Making it non-interactive

- Interactive protocol has soundness $^1/_N$, which isn't cryptographically small.
- Repeat τ times to get soundness $^1/_{N^\tau}$.
- (Use the same bundles from step 1 in all repetitions.)
- Apply the Fiat–Shamir transform to make it non-interactive:
    - Generate challenges in step 2 and 4 by hashing all previous commitments with a random oracle.

Lots of nice optimizations

- Linear operations involving secrets are basically free in MPC-in-the-head, so multiplying public A by secret A doesn’t add communication / increase size of proof
- Can generate lots of values from seeds and use seed trees to reduce size of proof
- Fast hashing using vectorized instructions

Defining security for proof of possession

Unforgeability:

- Hard to construct a valid proof of possession for an honest public key without the corresponding secret key

Zero knowledge:

- The proofs of possession leak no information about the secret key.

- Need to ensure the proof of possession composes nicely with the intended usage of the key
  - Zero knowledge shows the proof doesn't undermine the scheme
  - Need to extend unforgeability:
    - Use an "auxiliary secret key usage algorithm" in unforgeability experiment
    - Introduce a notion of KEM simulatability which FO-based KEMs have

Uniqueness of small FrodoKEM solutions

Recall high-level idea:

1. Generate and commit to many allegedly small values for S and E
2. Reveal some of them to prove they're small

This doesn't prove that all the unrevealed values are small, only most of them with high probability

- We prove a lemma upper-bounding the probability that a second FrodoKEM solution exists with mostly small solutions
- Choose number of bundles to audit to ensure no other mostly small secret key exists
- So proving possession of a mostly small solution implies proving possession of the true secret key
- Similar result for Kyber

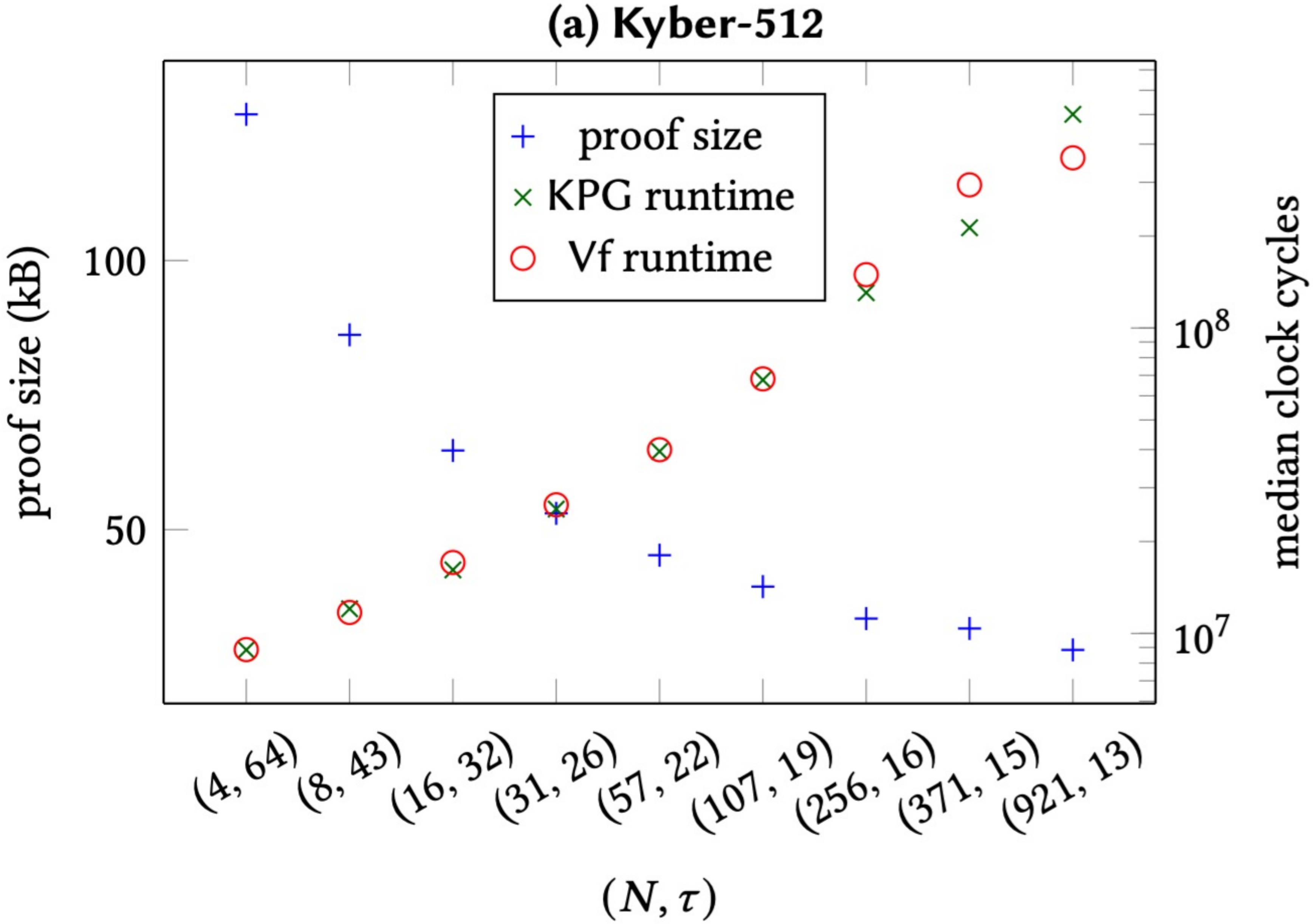
Performance trade-offs

52.9 kB / 0.01s

33.4 kB / 0.03s

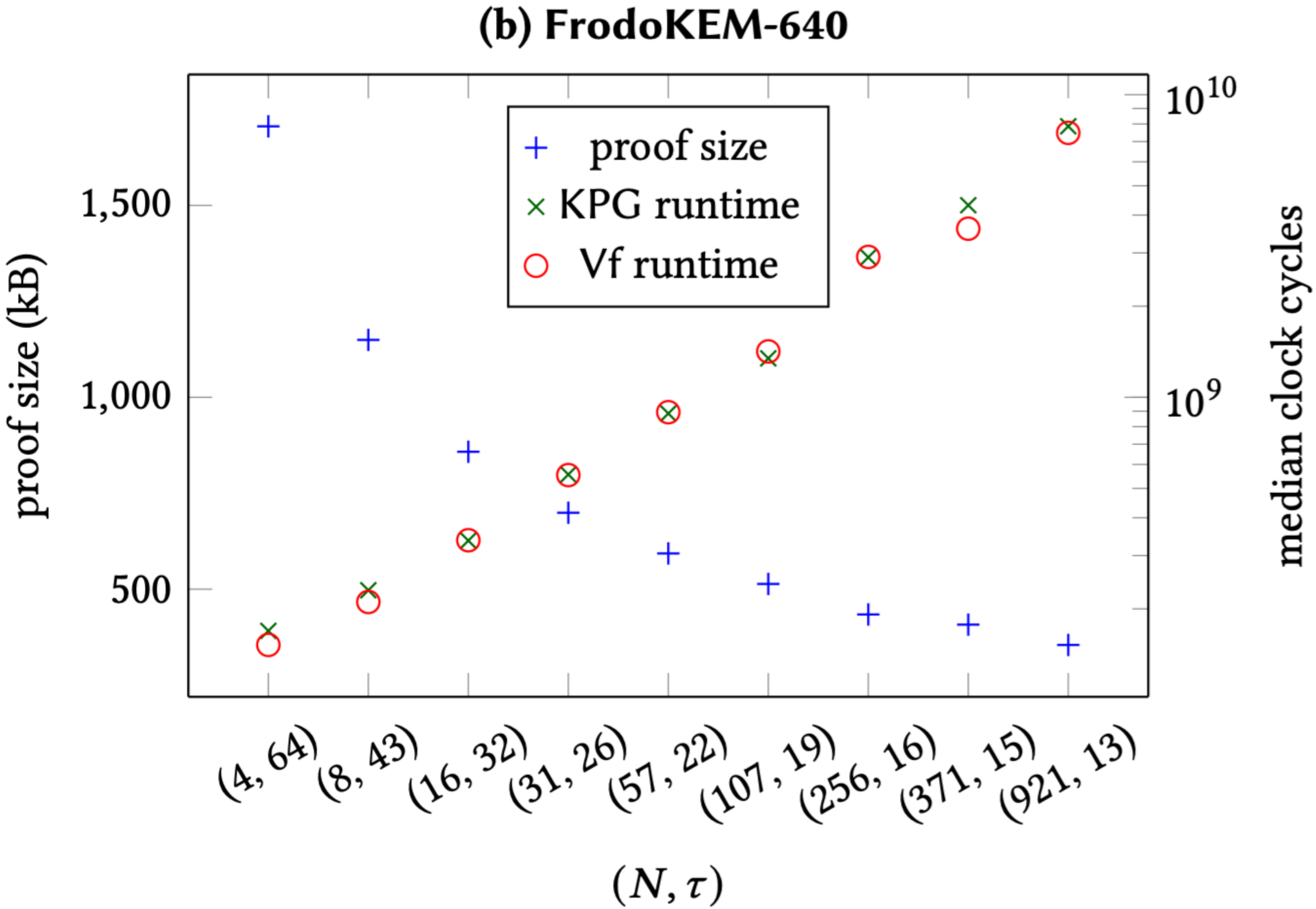
25.6 kB / 0.1s

17.8 kB / 3.8s



Intel Core i7-8565U CPU @ 4.6 GHz, gcc 11.2.0

Performance trade-offs



650 kB / 0.1s

402 kB / 0.6s

303 kB / 2.6s

203 kB / 85.6s

Intel Core i7-8565U CPU @ 4.6 GHz, gcc 11.2.0

Summary of verifiable generation

Verifiable generation with MPC-in-the-head yields reasonable proof sizes and runtimes for both FrodoKEM and Kyber at all security levels

- Smallest sizes can be competitive with direct lattice-based ZK constructions without needing to embed in a larger LWE instance with different parameters
- Order of magnitude smaller than previous MPC-in-the-head approaches

Recent results for KEMTLS

Douglas Stebila

University of Waterloo

KEMTLS

Implicitly authenticated TLS without handshake signatures using KEMs

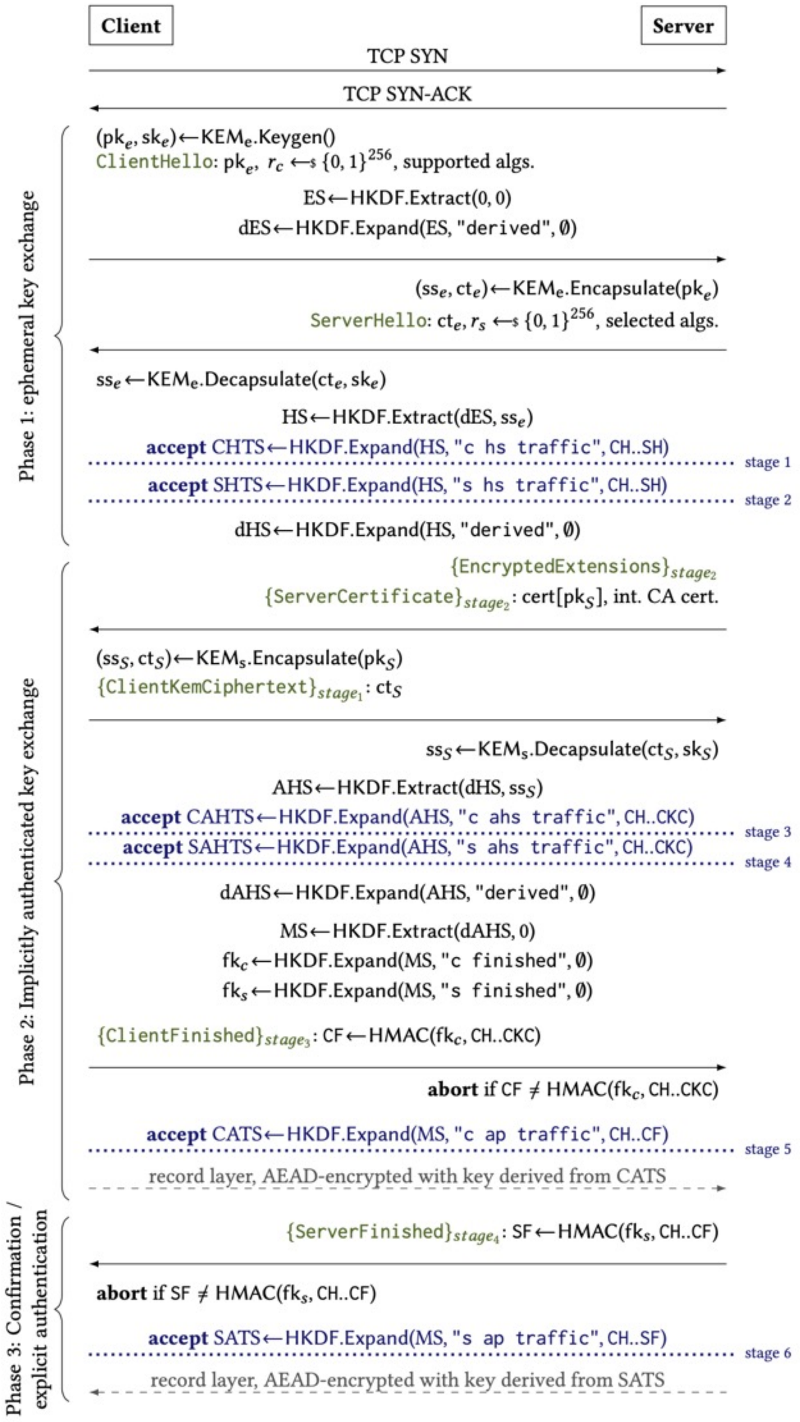
- Saves bytes on the wire, server cycles
- Variants for client authentication and pre-distributed public keys

1. KEMTLS design and performance
2. Pre-distributed public keys for faster client authentication
3. Proving KEMTLS manually and with Tamarin
   - Two Tamarin models with different levels of granularity
4. Certificate lifecycle for KEM public keys
   - Proof of possession via verifiable generation using MPC-in-the-head

https://www.douglas.stebila.ca/research
https://eprint.iacr.org/2020/534 • https://eprint.iacr.org/2021/779
https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00
https://github.com/thomwiggers/TLS13Tamarin • https://github.com/dstebila/KEMTLS-Tamarin/

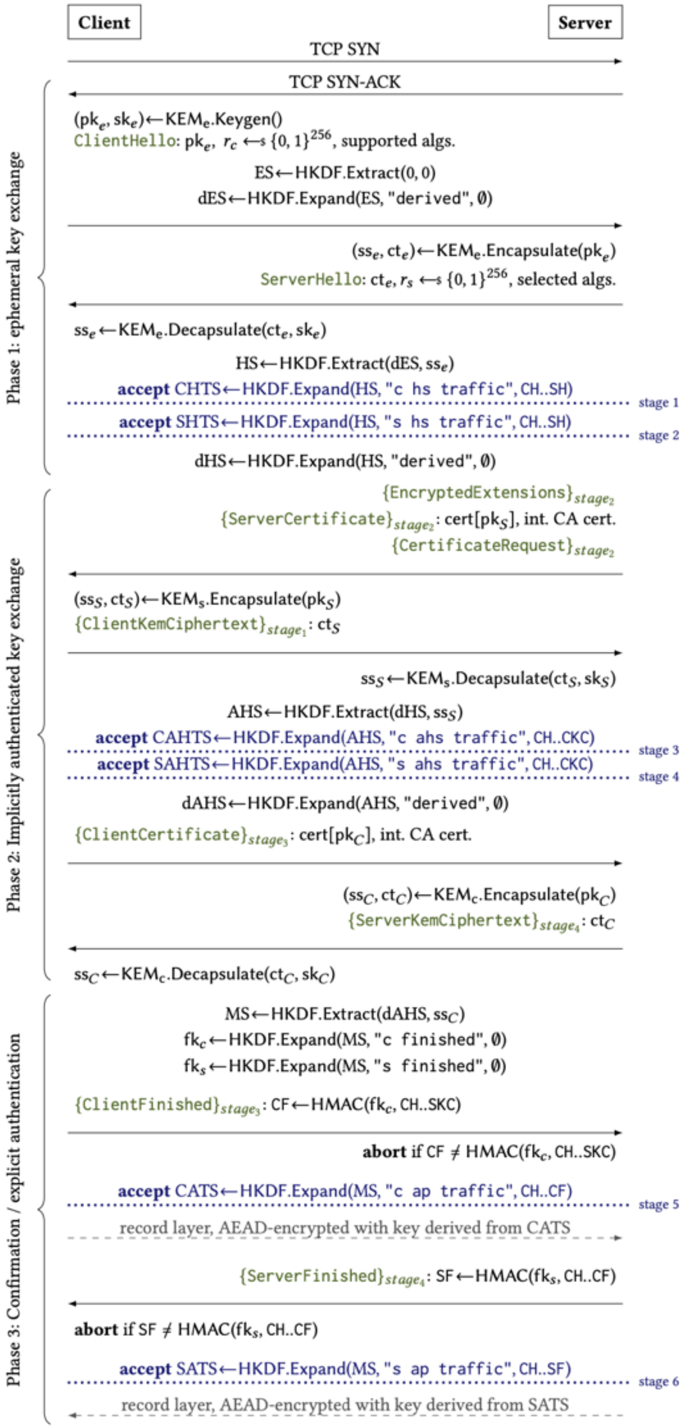
Appendix

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	2829
TLS 1.3 (Signed KEX)	**Min. incl. int. CA cert.**	SFXR	SIKE 433	Falcon 690	Falcon 897	$XMSS_s^{MT}$ 979	**2999**	$XMSS_s^{MT}$ 32	Rainbow 66	**3097**	Rainbow 161600	5378
TLS 1.3 (Signed KEX)	**Min. excl. int. CA cert.**	SFRR	SIKE 433	Falcon 690	Falcon 897	Rainbow 66	**2086**	Rainbow 60192	Rainbow 66	**62344**	Rainbow 60192	64693
TLS 1.3 (Signed KEX)	**Assumption: MLWE+MSIS**	KDDD	Kyber 1568	Dilithium 2420	Dilithium 1312	Dilithium 2420	**7720**	Dilithium 1312	Dilithium 2420	**11452**	Dilithium 1312	12639
TLS 1.3 (Signed KEX)	**Assumption: NTRU**	NFFF	NTRU 1398	Falcon 690	Falcon 897	Falcon 690	**3675**	Falcon 897	Falcon 690	**5262**	Falcon 897	6524
KEMTLS	**Min. incl. int. CA cert.**	SSXR	SIKE 433	SIKE 236	SIKE 197	$XMSS_s^{MT}$ 979	**1845**	$XMSS_s^{MT}$ 32	Rainbow 66	**1943**	Rainbow 60192	4252
KEMTLS	**Min. excl. int. CA cert.**	SSRR	SIKE 433	SIKE 236	SIKE 197	Rainbow 66	**932**	Rainbow 60192	Rainbow 66	**61190**	Rainbow 60192	63568
KEMTLS	**Assumption: MLWE+MSIS**	KKDD	Kyber 1568	Kyber 768	Kyber 800	Dilithium 2420	**5556**	Dilithium 1312	Dilithium 2420	**9288**	Dilithium 1312	10471
KEMTLS	**Assumption: NTRU**	NNFF	NTRU 1398	NTRU 699	NTRU 699	Falcon 690	**3486**	Falcon 897	Falcon 690	**5073**	Falcon 897	6359

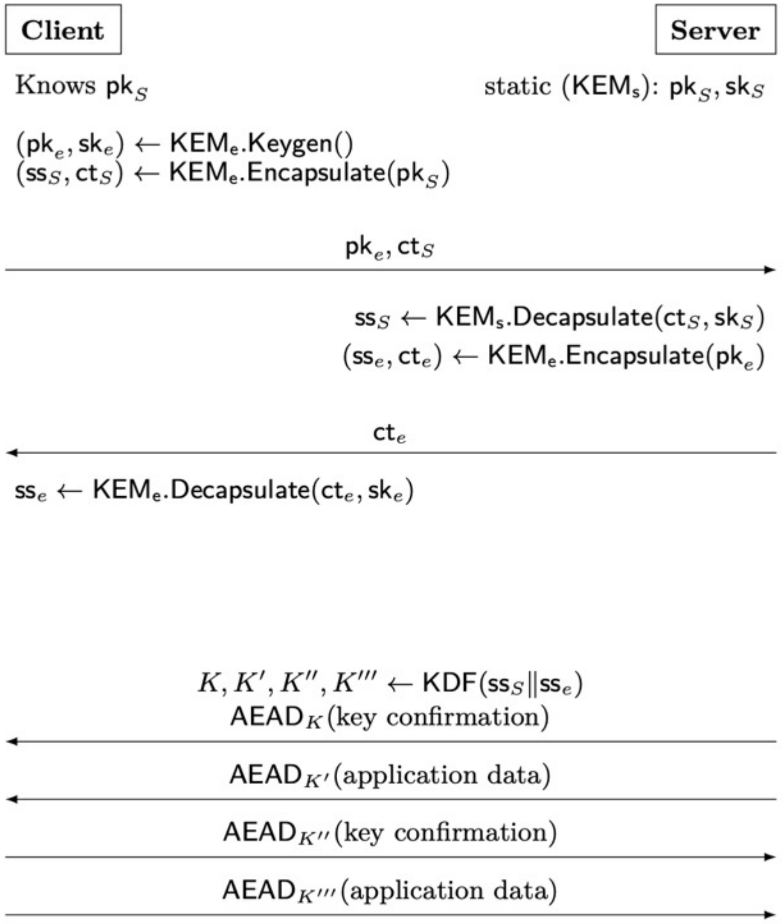
TLS 1.3 and KEMTLS crypto & handshake time

		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	**97.7**	35.5	66.5	**97.7**	35.5	397.3	**593.4**	201.4	398.3	**594.5**	202.4
	SFXR	11.860	4.410	12.051	4.410	80.1	**111.3**	49.2	80.4	**111.5**	49.4	417.5	**615.0**	218.9	417.4	**614.9**	219.1
	SFRR	6.061	4.410	6.251	4.410	65.5	**96.7**	34.5	131.4	**162.6**	100.4	398.3	**594.6**	201.8	1846.8	**2244.5**	1578.7
	KDDD	0.059	0.072	0.081	0.072	63.8	**95.1**	32.9	64.1	**95.4**	33.2	405.1	**602.3**	208.3	410.3	**609.8**	212.8
	NFFF	0.138	0.241	0.180	0.241	64.8	**96.0**	33.8	65.1	**96.4**	34.2	397.8	**593.9**	201.2	399.8	**596.0**	203.2
KEMTLS	**SSXR**	15.998	7.173	16.188	7.173	84.5	**124.6**	62.5	84.3	**124.4**	62.3	417.5	**625.8**	232.5	417.6	**625.8**	232.4
	SSRR	10.198	7.173	10.388	7.173	75.5	**116.3**	54.2	140.3	**182.3**	120.1	408.5	**616.5**	223.5	1684.2	**2091.6**	1280.4
	KKDD	0.048	0.017	0.070	0.017	63.3	**94.8**	32.6	63.7	**95.2**	32.9	397.3	**594.4**	201.6	434.7	**638.0**	235.4
	NNFF	0.107	0.021	0.149	0.021	63.4	**95.0**	32.7	63.7	**95.3**	33.0	395.9	**593.0**	200.1	397.6	**594.7**	201.9

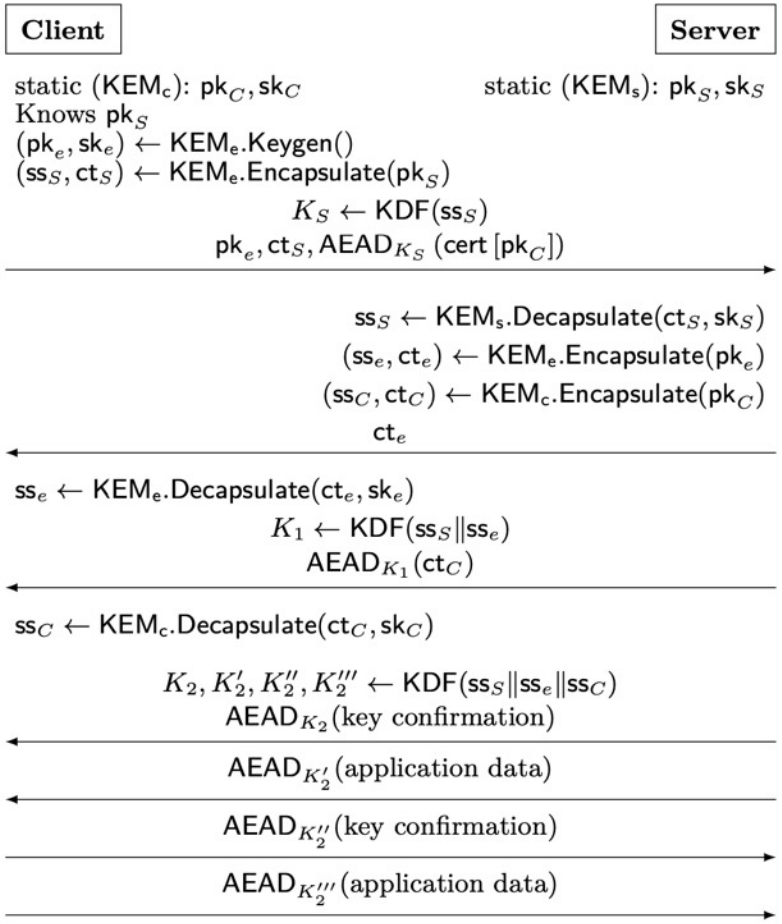
Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate CA certificate, D = root certificate.

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

KEMTLS-PDK overview

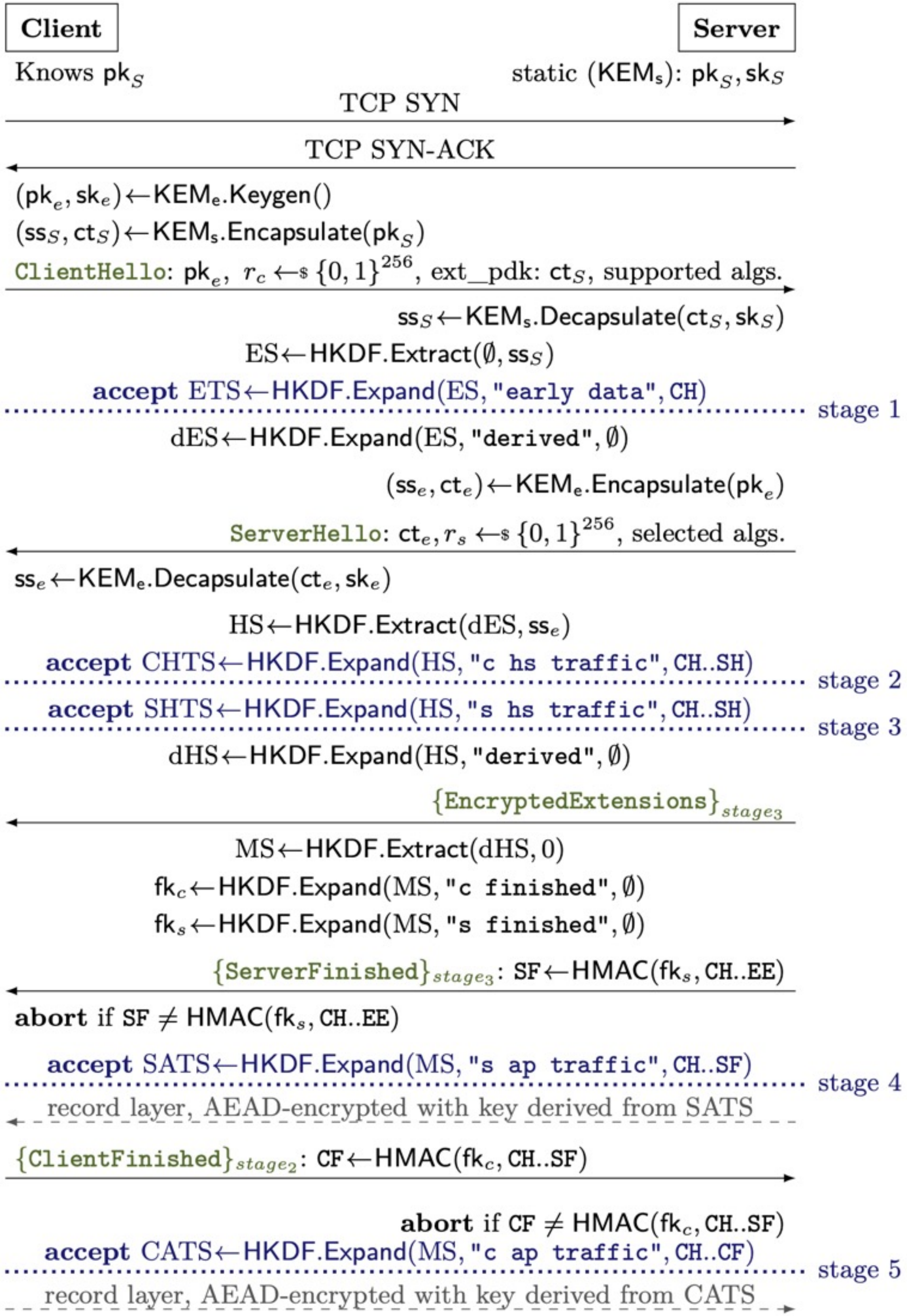


(a) Unilaterally authenticated

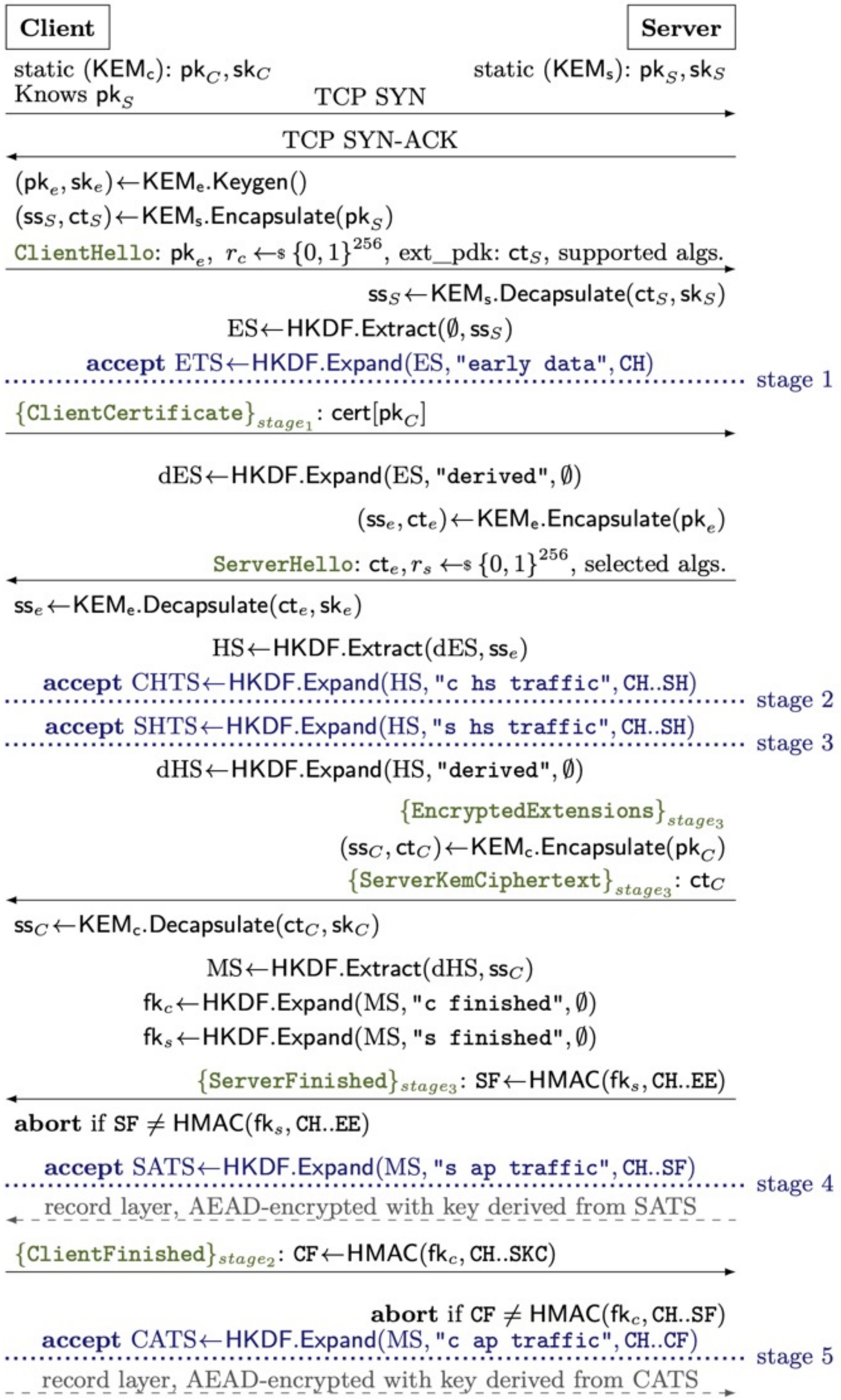


(b) With proactive client authentication

KEMTLS-PDK


Client
Server
Knows pk_S
static (KEM_s): pk_S, sk_S
TCP SYN
TCP SYN-ACK
$(pk_e, sk_e) \leftarrow KEM_e.Keygen()$
$(ss_S, ct_S) \leftarrow KEM_s.Encapsulate(pk_S)$
ClientHello: pk_e, $r_c \leftarrow_\$ \{0,1\}^{256}$, ext_pdk: ct_S, supported algs.
$ss_S \leftarrow KEM_s.Decapsulate(ct_S, sk_S)$
$ES \leftarrow HKDF.Extract(\emptyset, ss_S)$
accept $ETS \leftarrow HKDF.Expand(ES, \texttt{"early data"}, CH)$
stage 1
$dES \leftarrow HKDF.Expand(ES, \texttt{"derived"}, \emptyset)$
$(ss_e, ct_e) \leftarrow KEM_e.Encapsulate(pk_e)$
ServerHello: $ct_e, r_s \leftarrow_\$ \{0,1\}^{256}$, selected algs.
$ss_e \leftarrow KEM_e.Decapsulate(ct_e, sk_e)$
$HS \leftarrow HKDF.Extract(dES, ss_e)$
accept $CHTS \leftarrow HKDF.Expand(HS, \texttt{"c hs traffic"}, CH..SH)$
stage 2
accept $SHTS \leftarrow HKDF.Expand(HS, \texttt{"s hs traffic"}, CH..SH)$
stage 3
$dHS \leftarrow HKDF.Expand(HS, \texttt{"derived"}, \emptyset)$
$\{EncryptedExtensions\}_{stage_3}$
$MS \leftarrow HKDF.Extract(dHS, 0)$
$fk_c \leftarrow HKDF.Expand(MS, \texttt{"c finished"}, \emptyset)$
$fk_s \leftarrow HKDF.Expand(MS, \texttt{"s finished"}, \emptyset)$
$\{ServerFinished\}_{stage_3}$: $SF \leftarrow HMAC(fk_s, CH..EE)$
abort if $SF \neq HMAC(fk_s, CH..EE)$
accept $SATS \leftarrow HKDF.Expand(MS, \texttt{"s ap traffic"}, CH..SF)$
stage 4
record layer, AEAD-encrypted with key derived from SATS
$\{ClientFinished\}_{stage_2}$: $CF \leftarrow HMAC(fk_c, CH..SF)$
abort if $CF \neq HMAC(fk_c, CH..SF)$
accept $CATS \leftarrow HKDF.Expand(MS, \texttt{"c ap traffic"}, CH..CF)$
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
	MLWE/MSIS	95.2	**126.6**	126.6	598.3	**794.6**	794.6
	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
	Minimum	71.1	**102.7**	69.9	403.3	**602.0**	402.0
	MLWE/MSIS	64.5	**96.2**	63.9	400.1	**616.8**	399.5
	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
	Kyber	63.5	**94.7**	63.4	400.2	**596.5**	400.2
	NTRU	63.6	**94.9**	63.6	397.6	**593.8**	397.5
	SABER	63.6	**94.8**	63.5	399.4	**595.5**	399.3

	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

Tamarin runtimes for Approach 2

Lemma	KEMTLS			KEMTLS-PDK			**All 4**
	sauth	mutual	both	sauth	mutual	both	**variants**
reachable_*	0:01:17	0:01:20	0:04:32	0:01:46	0:01:36	0:04:40	0:13:25
attacker_works_*	0:00:17	0:00:46	0:01:16	0:00:17	0:00:23	0:00:53	0:12:04
match_*	0:01:02	0:01:22	0:02:55	0:00:55	0:01:14	0:02:46	0:09:53
sk_sec_nofs_client	0:00:05	0:00:07	0:00:16	0:00:05	0:00:05	0:00:14	0:00:41
sk_sec_nofs_server	0:00:05	0:00:06	0:00:12	0:00:05	0:00:06	0:00:14	0:00:40
sk_sec_wfs1	0:00:21	0:00:10	0:01:05	0:00:17	0:00:18	0:00:41	0:03:00
sk_sec_wfs2	0:00:36	0:00:28	0:01:30	0:00:28	0:00:22	0:01:23	0:24:28
sk_sec_fs	0:01:20	0:03:05	0:06:38	0:01:21	0:01:33	0:05:07	1:39:58
malicious_accept.	0:00:13	0:01:40	0:04:13	0:00:17	0:00:22	0:01:39	27:29:37
deniability (abbr.)	0:01:02	0:12:15	—	0:00:24	0:29:10	—	—
Total (excl. **den.**)	0:05:16	0:09:05	0:22:38	0:05:30	0:06:00	0:17:38	30:13:46

Proof of possession comparison

Scheme	Technique	**Regime 1**	**Regime 2**		Kyber512		Frodo640	
		Size	**Size**	**Time**	**Size**	**Time**	**Size**	**Time**
Proof of knowledge of secret key (and proof of verifiable decryption, denoted ◇)								
Stern-like [51]	ZKP from SIS	2.3 MB†	4.3 MB†					
[9]	MPCitH		4.1 MB	2.4 s			≥ 8.42 MB‡	
[20]	ZKP from RLWE & RSIS	384 kB†						
[13]	Σ-prot. for permuted-kernel	233 kB	444 kB					
Ligero [5]	zkSNARK from PCPs	157 kB†	200 kB†					
Aurora [11]	zkSNARK for R1CS	72 kB†	71 kB†					
[37]	ZKP from MLWE & MSIS	47 kB, 61 kB◇						
[56]	ZKP from MLWE & MSIS	47 kB*						
[57]	ZKP from MSIS & ext. MLWE	33 kB			43.6 kB◇			
[55]	ZKP from MLWE & MSIS	14 kB			19.0 kB◇			
Proof of verifiable generation								
Ours (31, 26)	MPCitH	251 kB	879 kB		52.9 kB	0.006 s	650 kB	0.12 s
Ours (256, 16)	MPCitH	155 kB	542 kB		33.4 kB	0.028 s	402 kB	0.63 s
Ours (1626, 12)	MPCitH	117 kB	407 kB		25.7 kB	0.109 s	302 kB	2.59 s
Ours (65536, 8)	MPCitH	79 kB	272 kB		17.8 kB	3.77 s	203 kB	85.6 s

Empty cell indicates estimates for parameter regime not available in the original paper or subsequent literature.

"Ours (N, τ)" denotes number N of MPC-in-the-head parties and number τ of MPC repetitions.

Regime 1: modulus $q \approx 2^{32}$, number of secret entries $\sigma = 2048$, ternary secrets $\{-1, 0, 1\}$, 128-bit security level.

Regime 2: modulus $q \approx 2^{61}$, number of secret entries $\sigma = 4096$, binary secrets $\{0, 1\}$, 128-bit security level.

Kyber512: modulus $q = 3329$, number of secret entries $\sigma = 1024$, secret $[0, \pm 2]$, 128-bit security level.

Frodo640: modulus $q = 2^{15}$, number of secret entries $\sigma = 10240$, secret $[0, \pm 12]$, 128-bit security level.

†: Estimates by Beullens [13]. ‡: Estimate by us, using edaBits [36] for comparisons.

*: Estimate by Lyubashevsky et al. [57]. ◇: For proof of verifiable decryption.

Runtime for a single-threaded implementation on Intel Core i7-8565U CPU running at up to 4.6 GHz, compiled with gcc 11.2.0.