

# Integrating post-quantum cryptography into real-world protocols

**Douglas Stebila**



<https://www.douglas.stebila.ca/research/presentations/>

# Outline

## Part 1: Existing protocol designs

- Classical + PQ key exchange
- Classical + PQ signatures
- Performance

## Part 2: Alternative protocol designs

- KEMTLS

# 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

$(pk_A, sk_A) \leftarrow \text{SIG.KeyGen}()$

obtain  $pk_B$

$x \leftarrow_s \{0, \dots, q-1\}$

$X \leftarrow g^x$



Bob

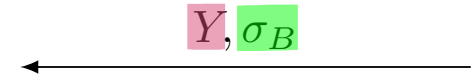
$(pk_B, sk_B) \leftarrow \text{SIG.KeyGen}()$

obtain  $pk_A$

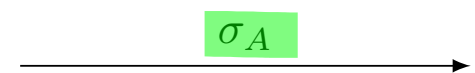
$y \leftarrow_s \{0, \dots, q-1\}$

$Y \leftarrow g^y$

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

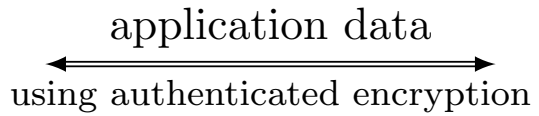


$\sigma_A \leftarrow \text{SIG.Sign}(sk_A, A||B||X||Y)$



$k \leftarrow H(sid, Y^x)$

$k \leftarrow H(sid, X^y)$



# Implicitly authenticated key exchange: Double-DH

Alice

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

$$pk_A \leftarrow g^{sk_A}$$

obtain  $pk_B$

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

$$X \leftarrow g^x$$

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

Bob

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

$$pk_B \leftarrow g^{sk_B}$$

obtain  $pk_A$

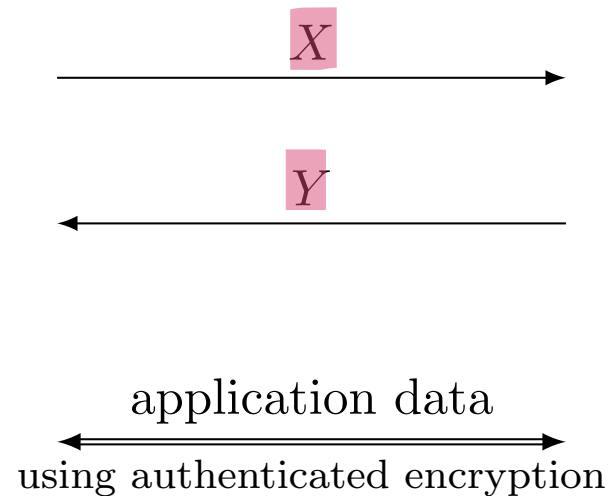
$$y \leftarrow_{\$} \{0, \dots, q-1\}$$

$$Y \leftarrow g^y$$

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

Static DH

Ephemeral DH



# 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
<b>Dilithium</b>	<b>Lattice-based (MLWE/MSIS)</b>	<b>1,184</b>	<b>2,044</b>
<b>Falcon</b>	<b>Lattice-based (NTRU)</b>	<b>897</b>	<b>690</b>
<b>XMSS</b>	<b>Hash-based</b>	<b>32</b>	<b>979</b>
<b>Rainbow</b>	<b>Multi-variate</b>	<b>60,192</b>	<b>66</b>

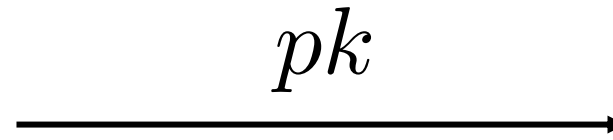
# Solution

use  
post-quantum KEMs  
for authentication

# Key encapsulation mechanisms (KEMs)

An abstraction of Diffie–Hellman key exchange

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



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



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

Signature scheme		Public key (bytes)	Signature (bytes)
RSA-2048	Factoring	272	256
Elliptic curves	Elliptic curve discrete logarithm	32	32
<b>Dilithium</b>	<b>Lattice-based (MLWE/MSIS)</b>	<b>1,184</b>	<b>2,044</b>
<b>Falcon</b>	<b>Lattice-based (NTRU)</b>	<b>897</b>	<b>690</b>
<b>XMSS</b>	<b>Hash-based</b>	<b>32</b>	<b>979</b>
<b>Rainbow</b>	<b>Multi-variate</b>	<b>60,192</b>	<b>66</b>

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

# Implicitly authenticated KEX is not new

## In theory

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

## In practice

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

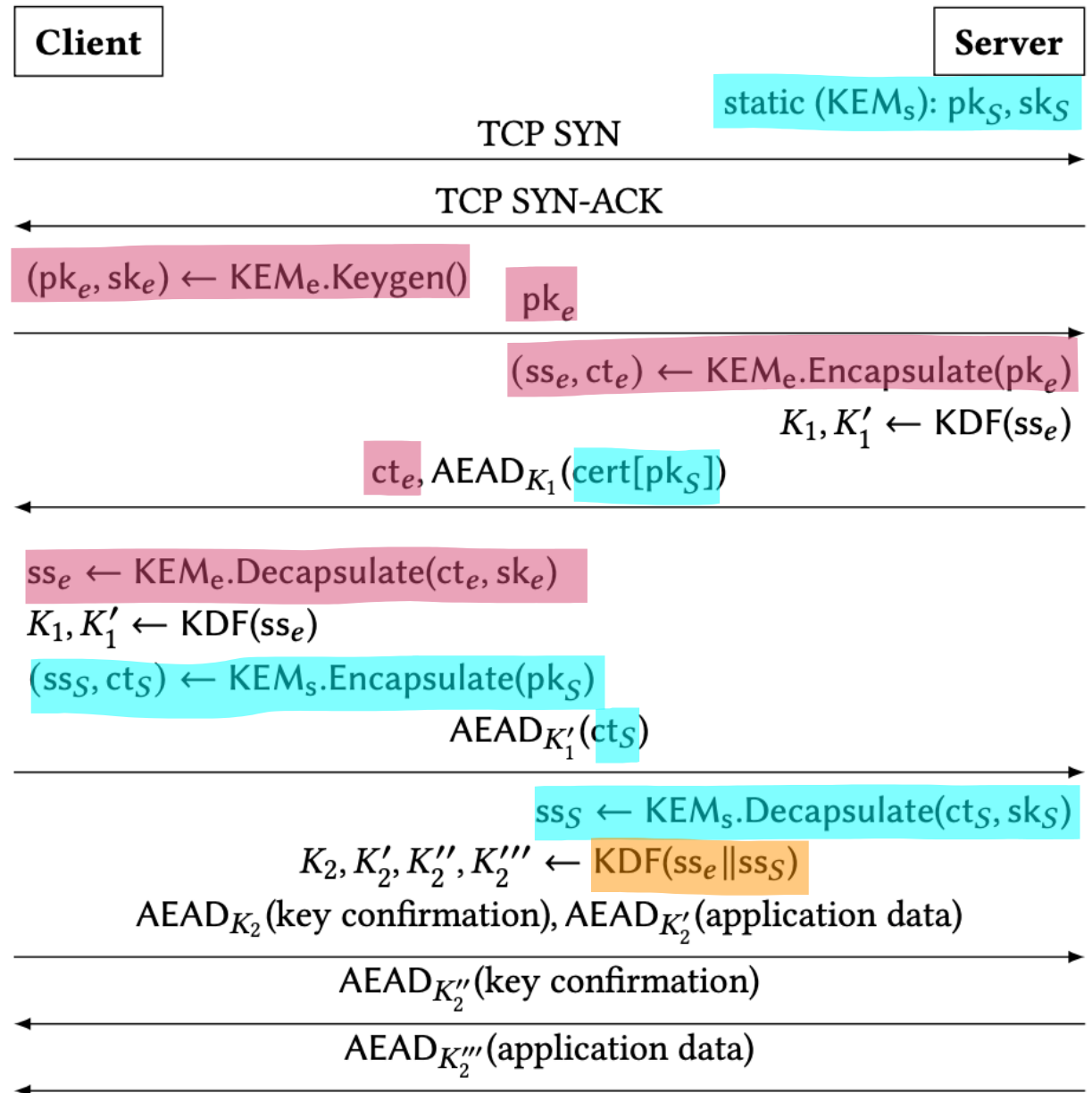


# KEMTLS handshake

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

## Signature scheme for intermediate CA

- Want small public key + small signature

## KEM for authenticated key exchange

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

## Signature scheme for root CA

- Want small signature

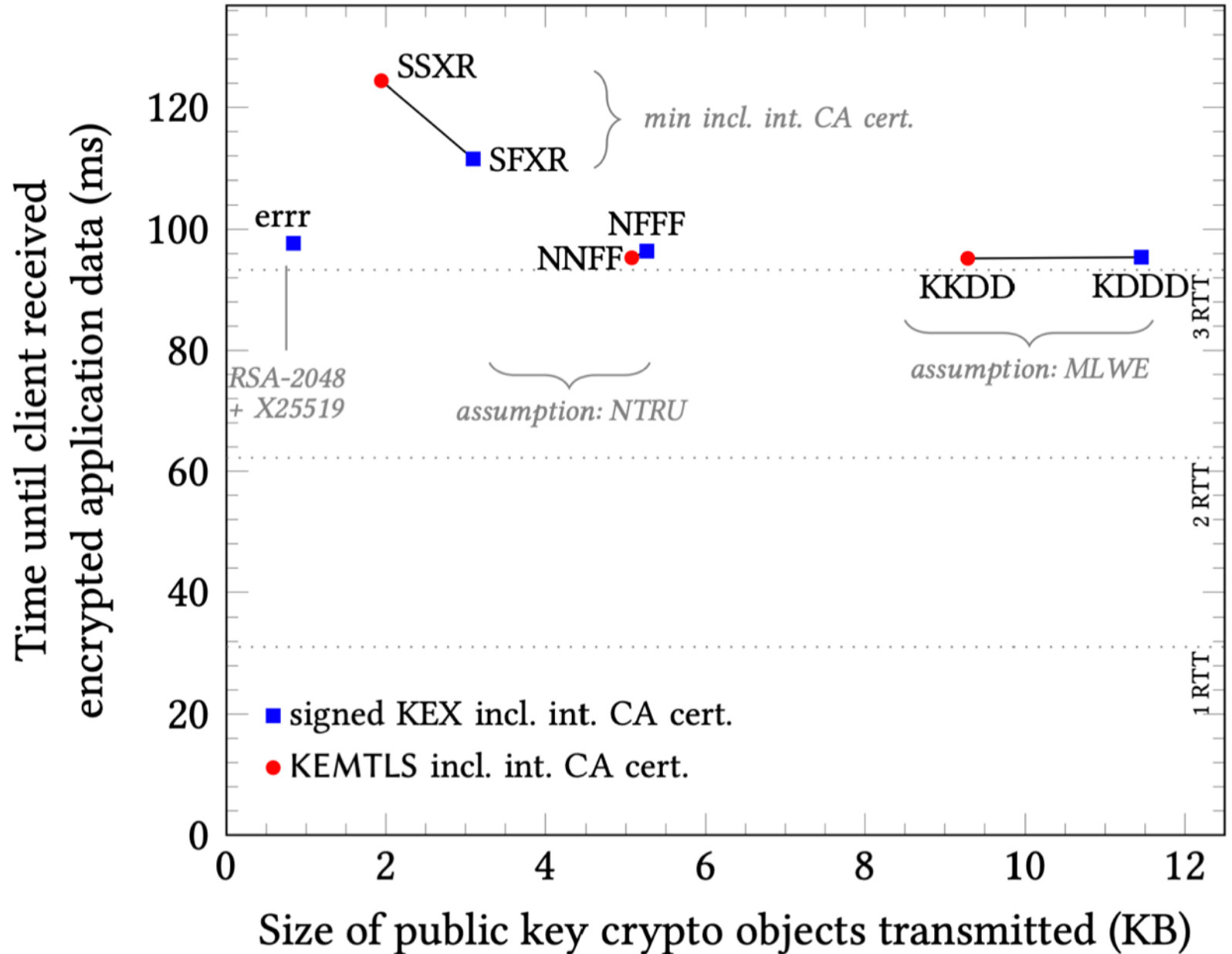
# 4 scenarios

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

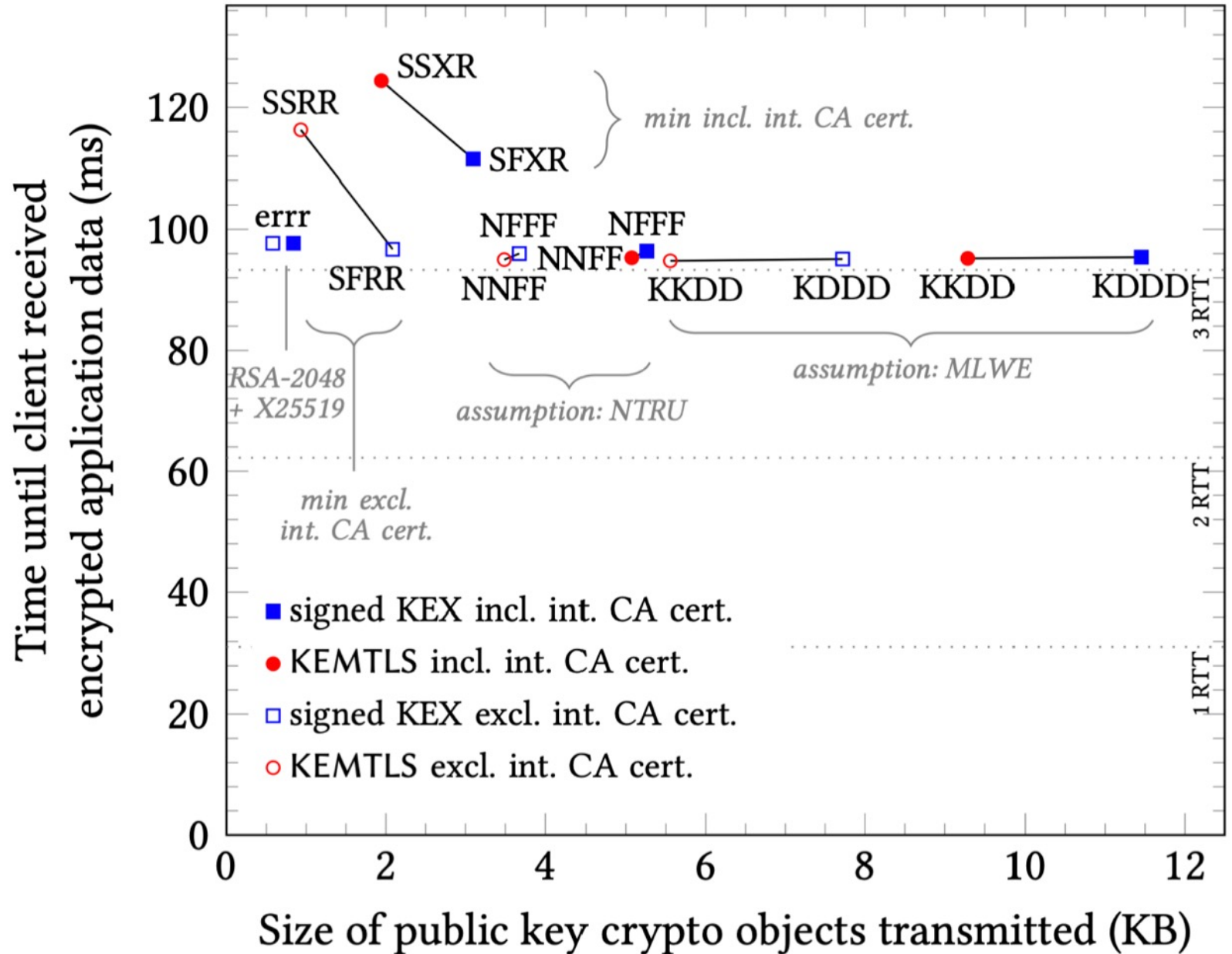
Algorithms: (all level 1)  
 Dilithium,  
 eCDH X25519,  
 Falcon,  
 Kyber,  
 NTRU,  
 Rainbow,  
 rSA-2048,  
 SIKE,  
 XMSS'



# 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'



# KEMTLS benefits

- Size-optimized KEMTLS requires  $< \frac{1}{2}$  communication of size-optimized PQ signed-KEM
- Speed-optimized KEMTLS uses 90% fewer server CPU cycles and still reduces communication
  - NTRU KEX (27  $\mu$ s) 10x faster than Falcon signing (254  $\mu$ s)
- 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

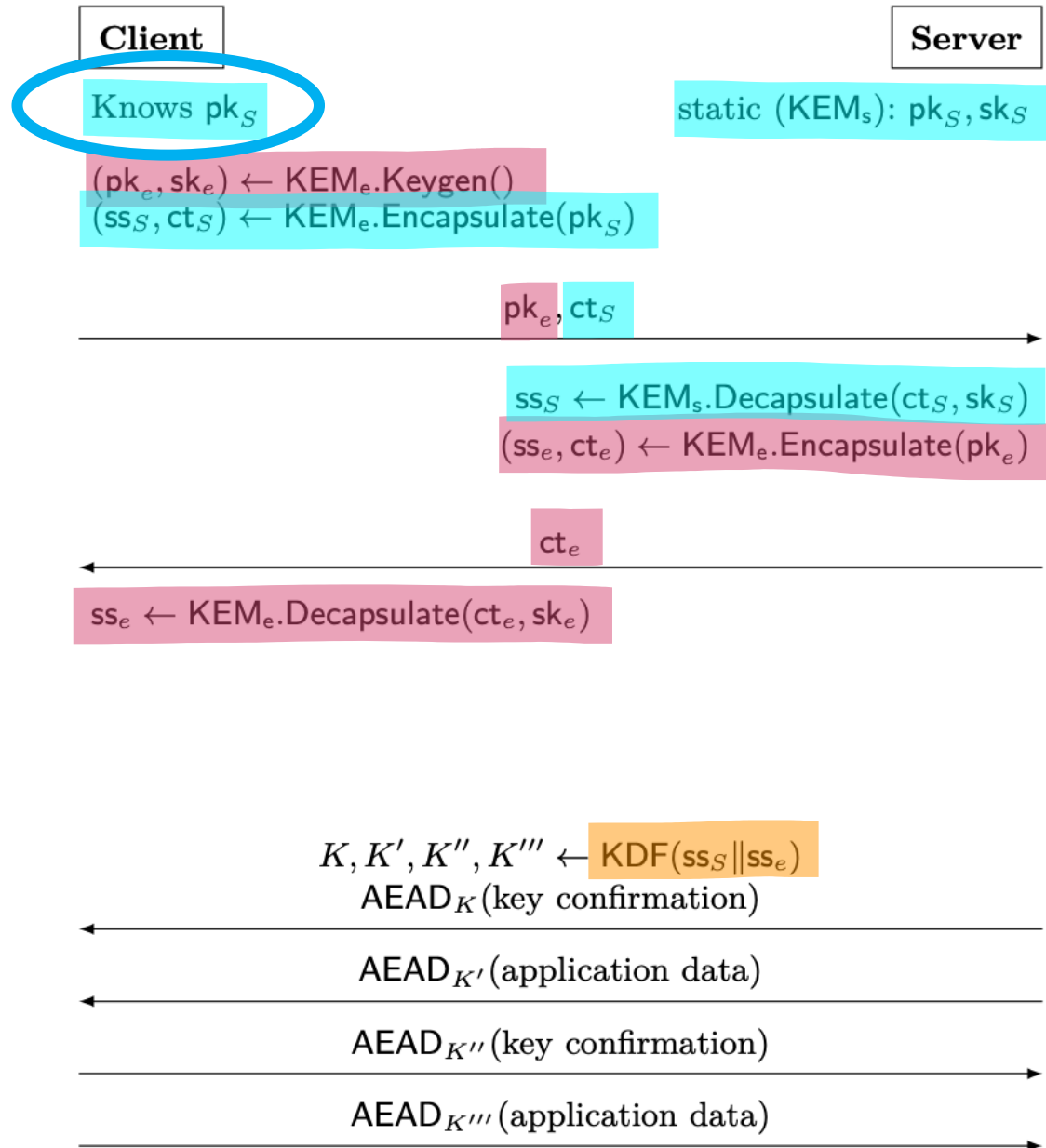
**Get a 1-RTT,  
TLS 1.3-shape  
handshake with  
implicit authentication**

# KEMTLS-PDK handshake server-only auth.

KEM for  
ephemeral key exchange

KEM for  
server-to-client  
authenticated key exchange

Combine shared secrets



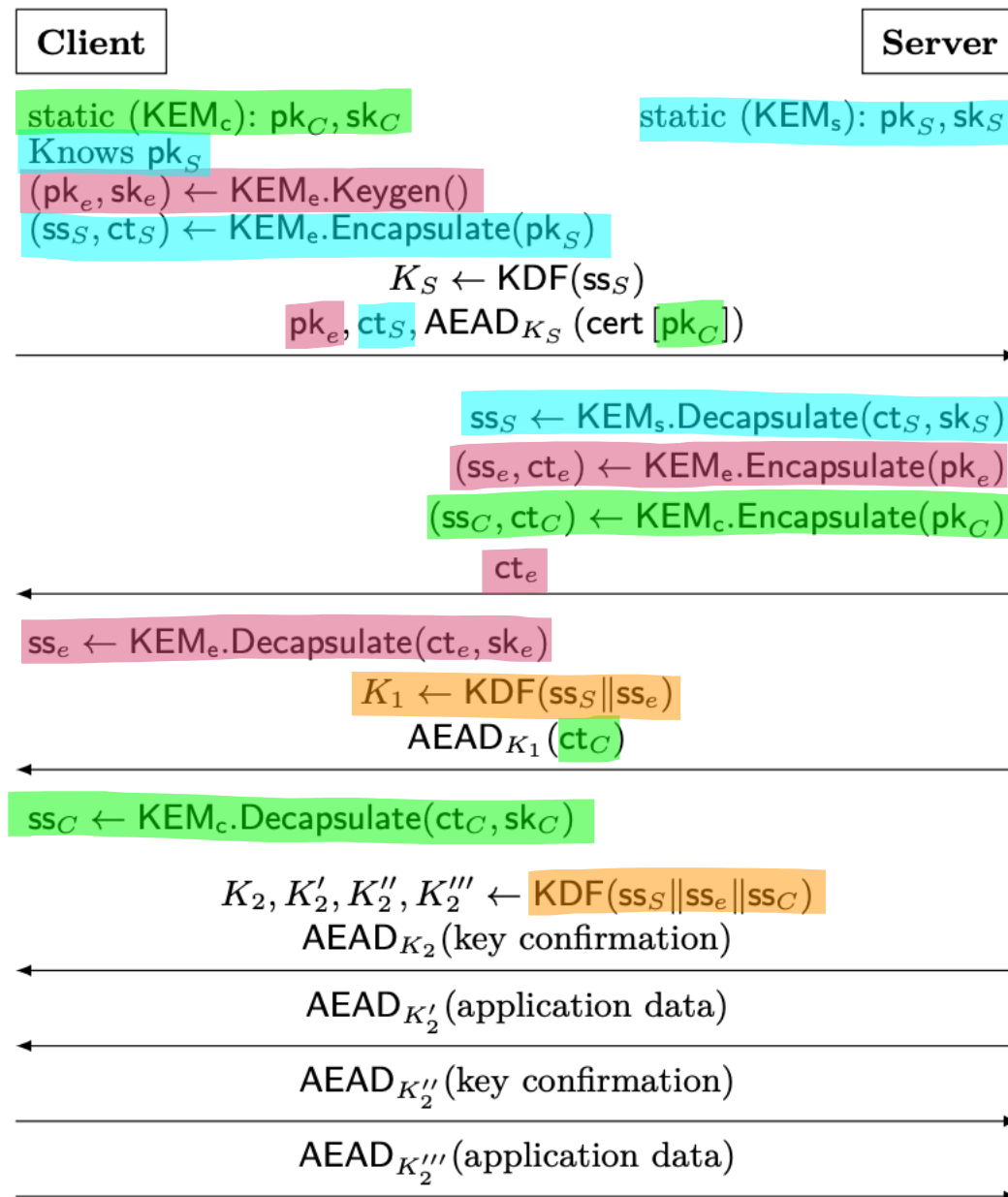
# KEMTLS-PDK handshake mutual auth

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

# 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>

Sofia 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



# 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

# 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

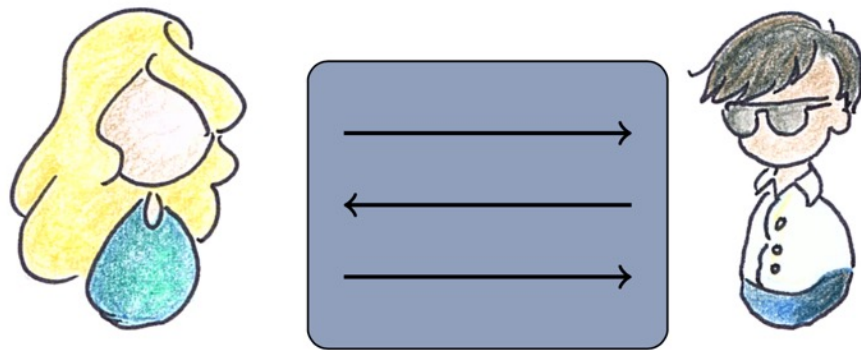
## Formal verification

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

# Provable security approach to analyzing key exchange protocols

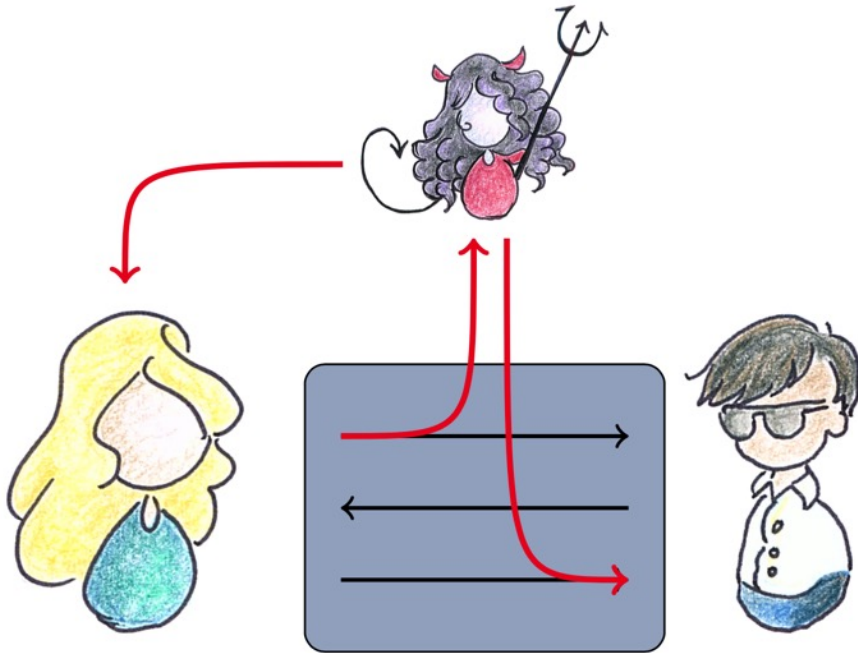
## 1. describe abstract protocol





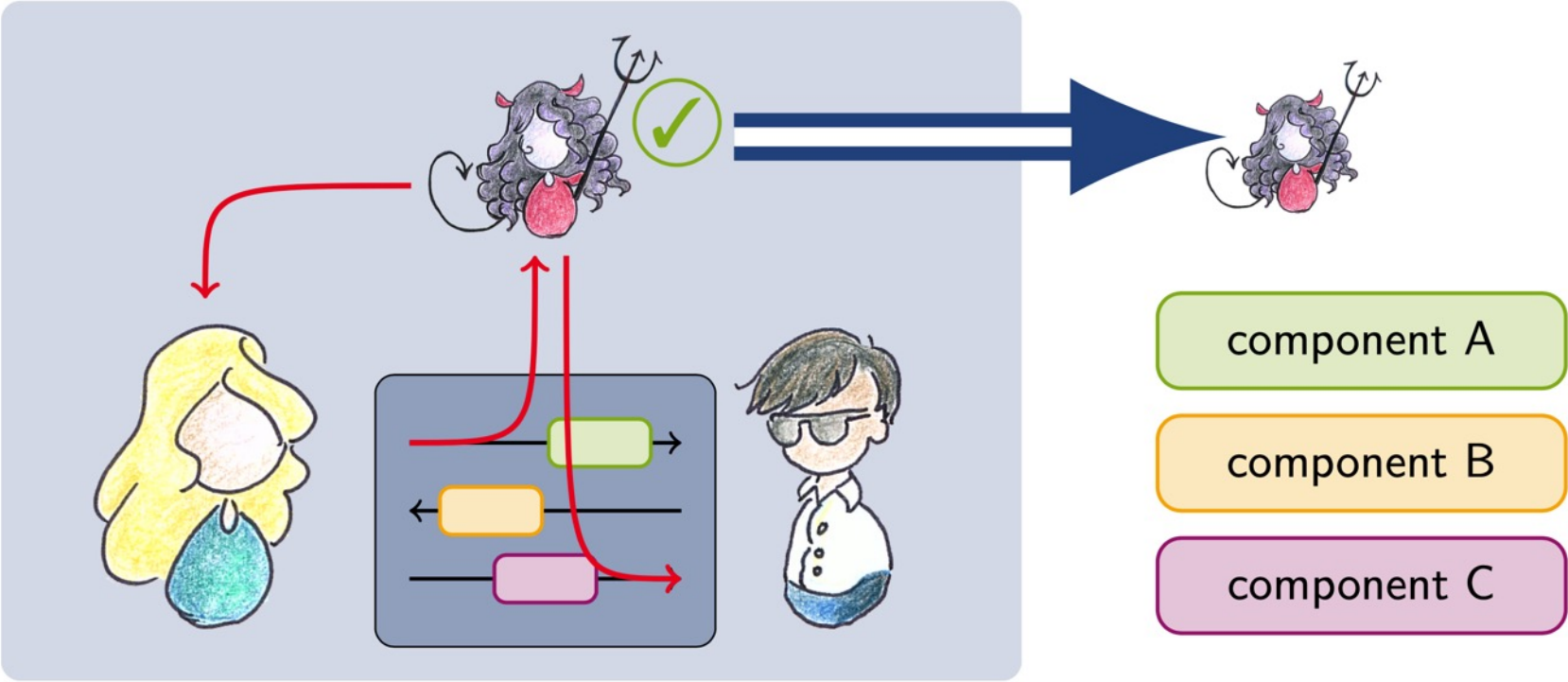
# Provable security approach to analyzing key exchange protocols

1. describe abstract protocol
2. define security



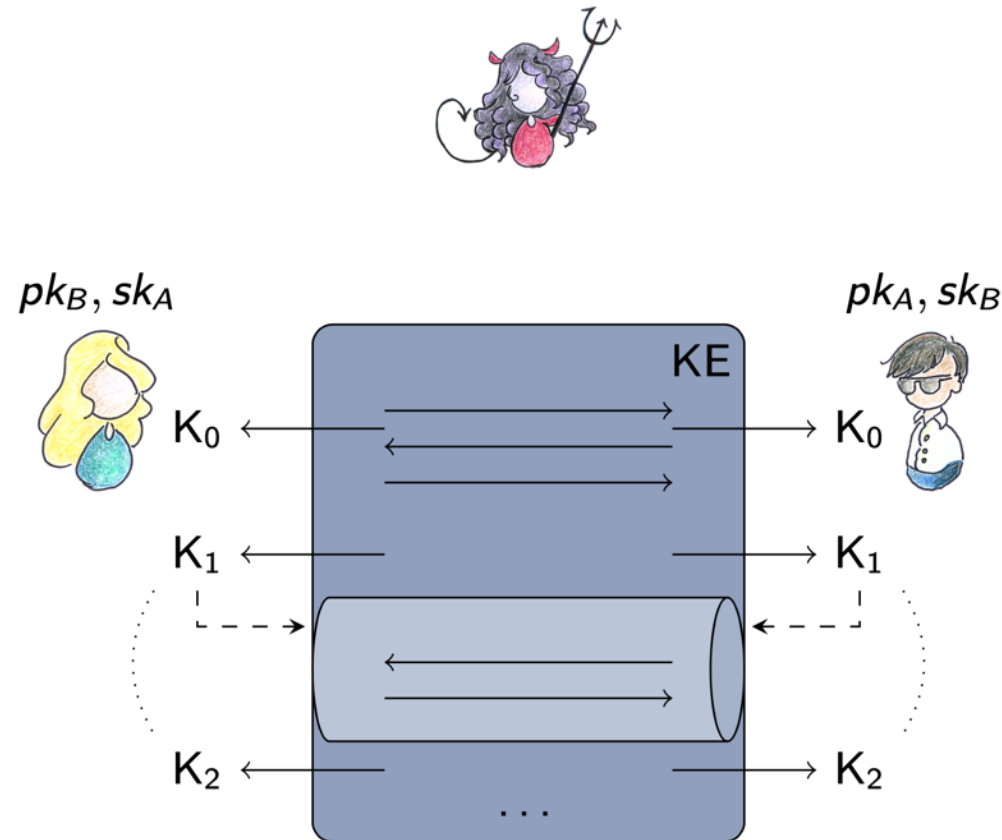
# Provable security approach to analyzing key exchange protocols

- 1. describe abstract protocol
- 2. define security
- 3. reduce to assumptions



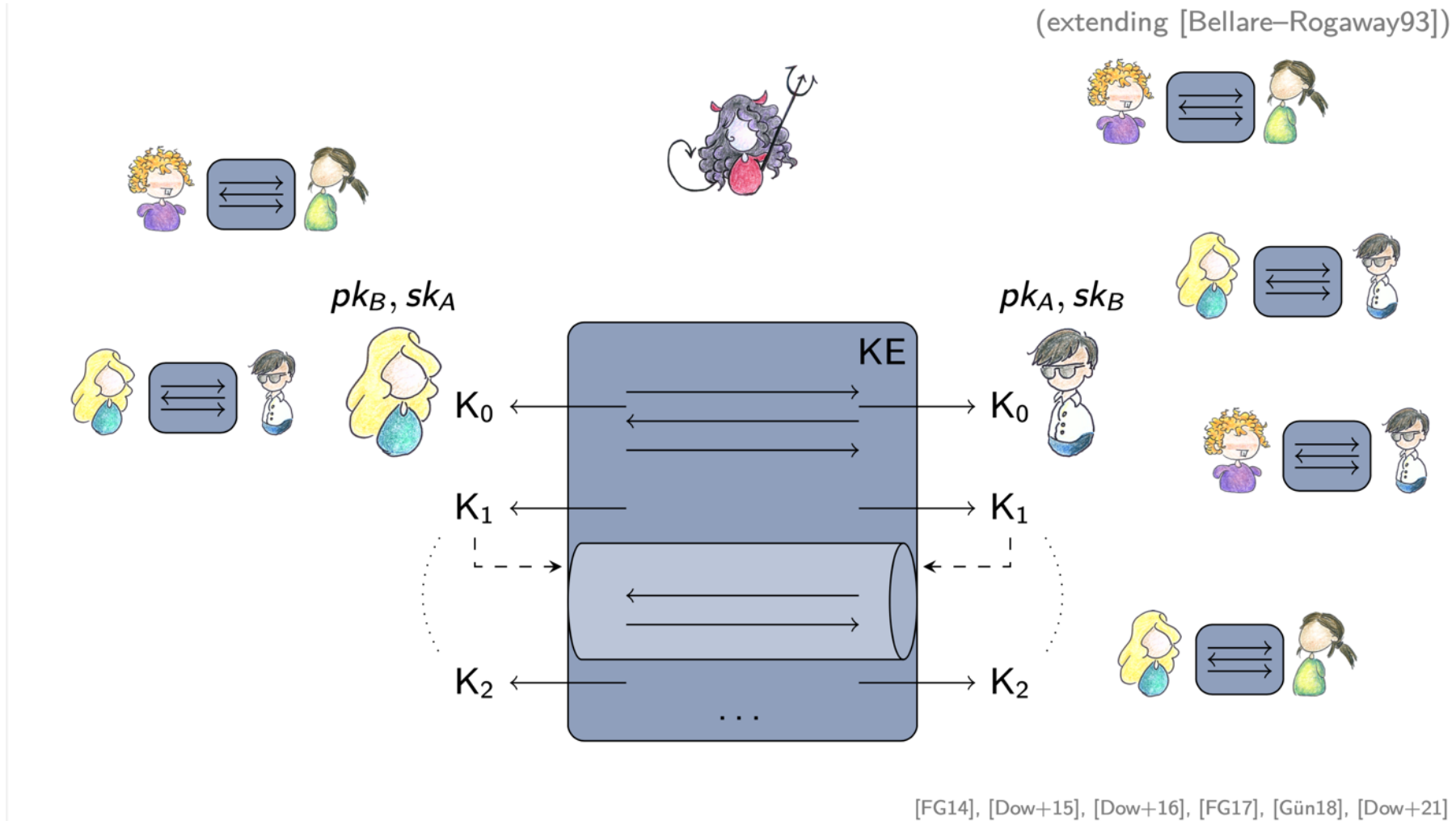
# Multi-stage key exchange security

(extending [Bellare–Rogaway93])



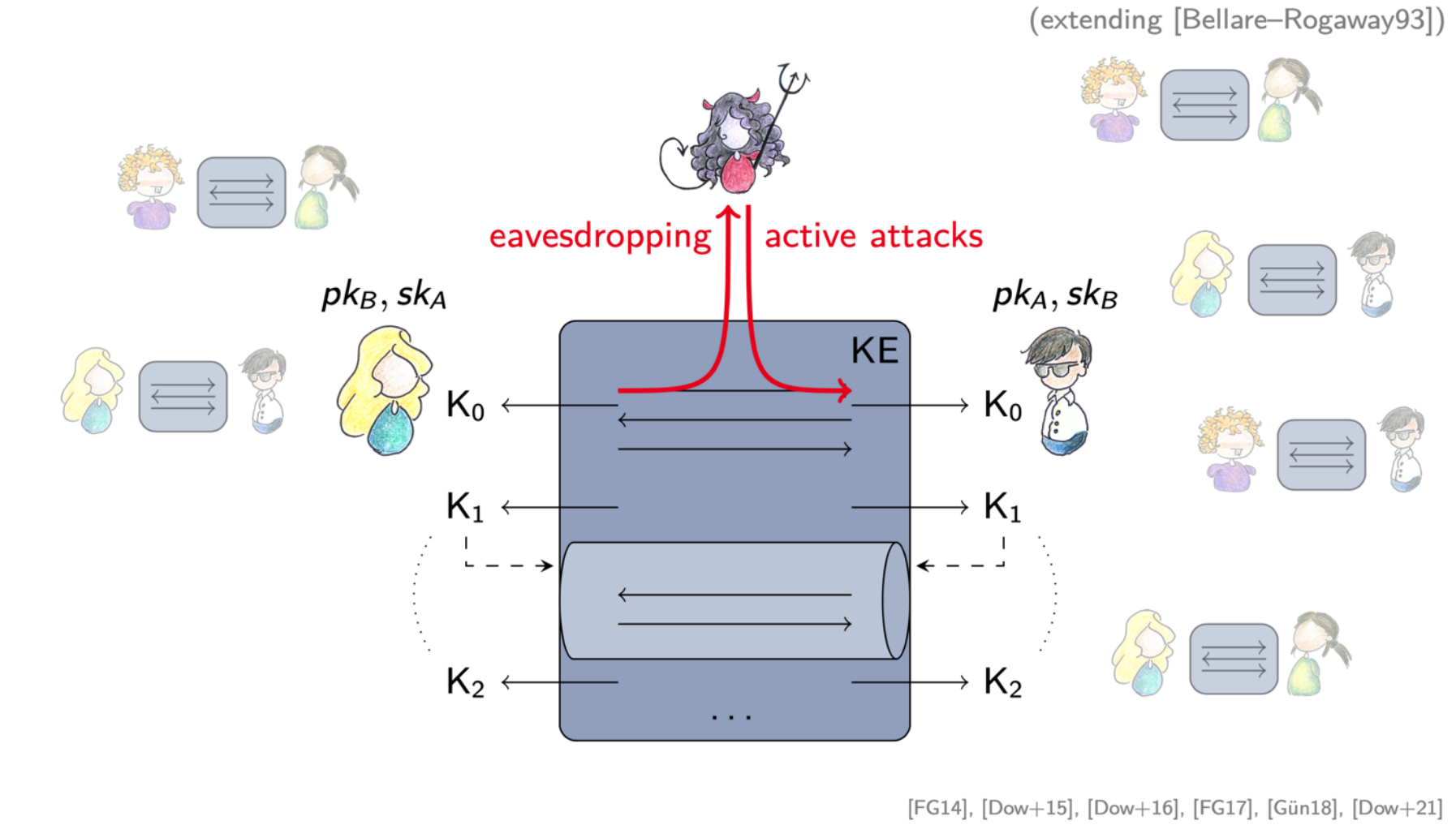
[FG14], [Dow+15], [Dow+16], [FG17], [Gün18], [Dow+21]

# Multi-stage key exchange security



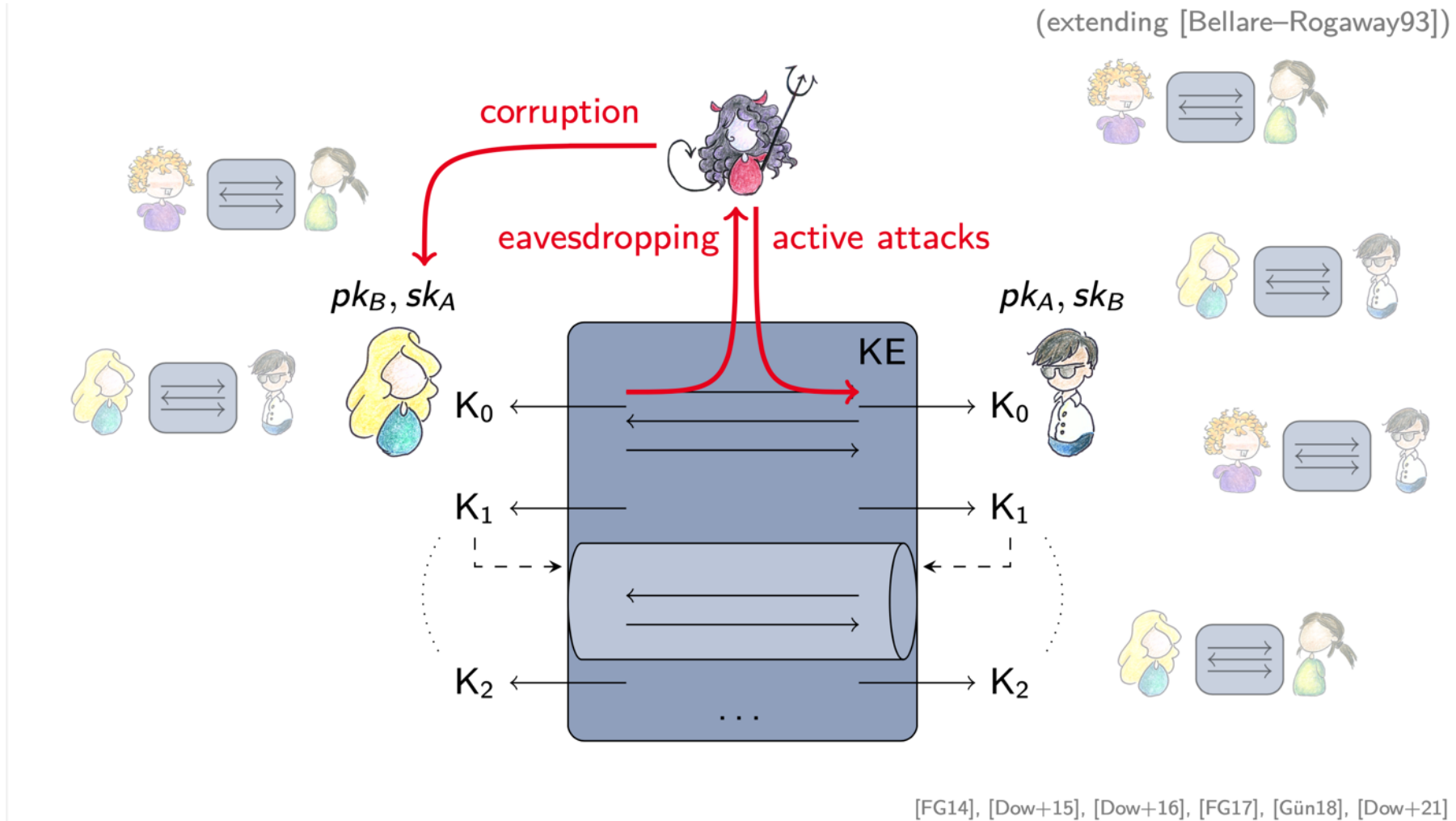
[FG14], [Dow+15], [Dow+16], [FG17], [Gün18], [Dow+21]

# Multi-stage key exchange security

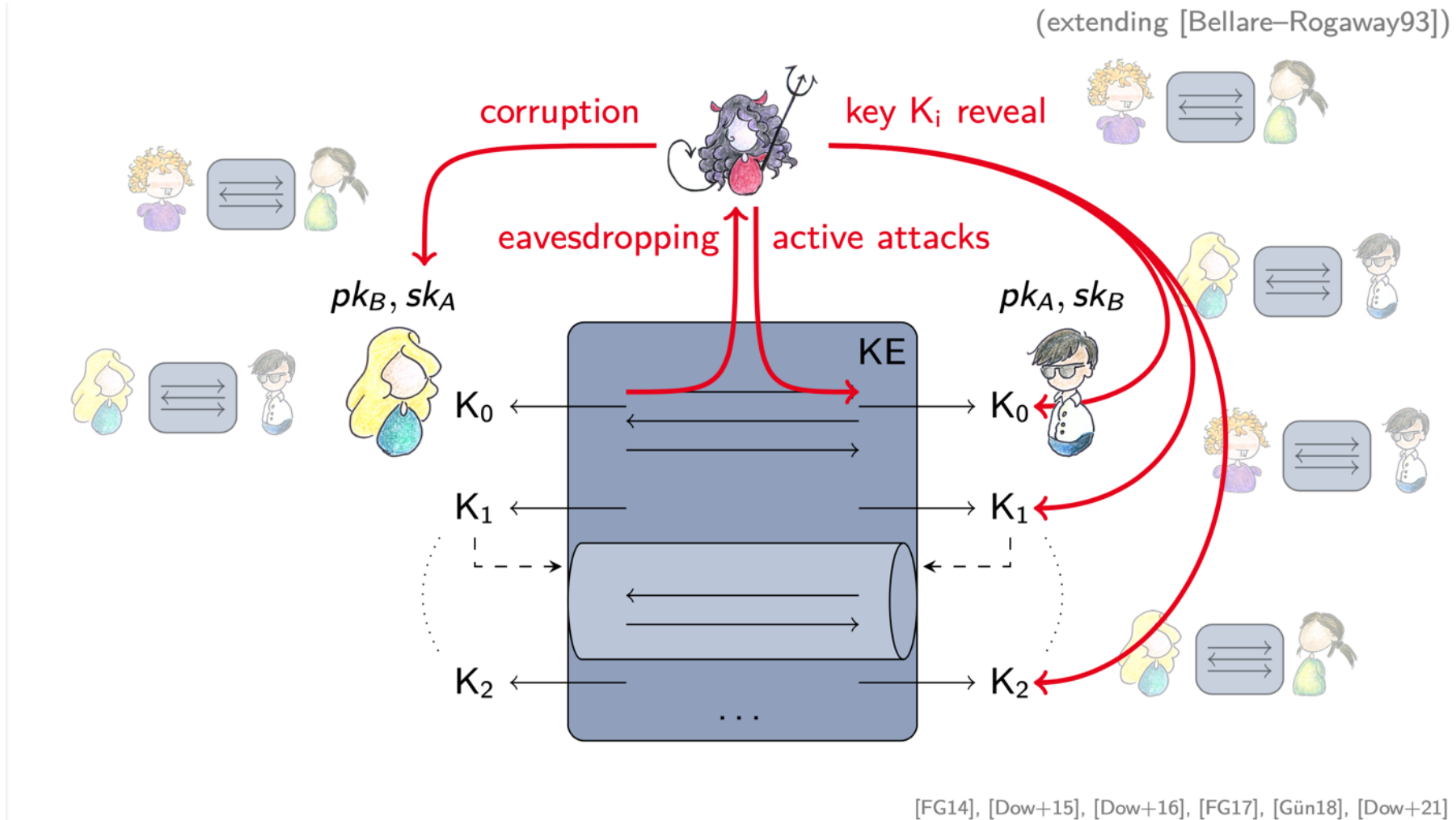


[FG14], [Dow+15], [Dow+16], [FG17], [Gün18], [Dow+21]

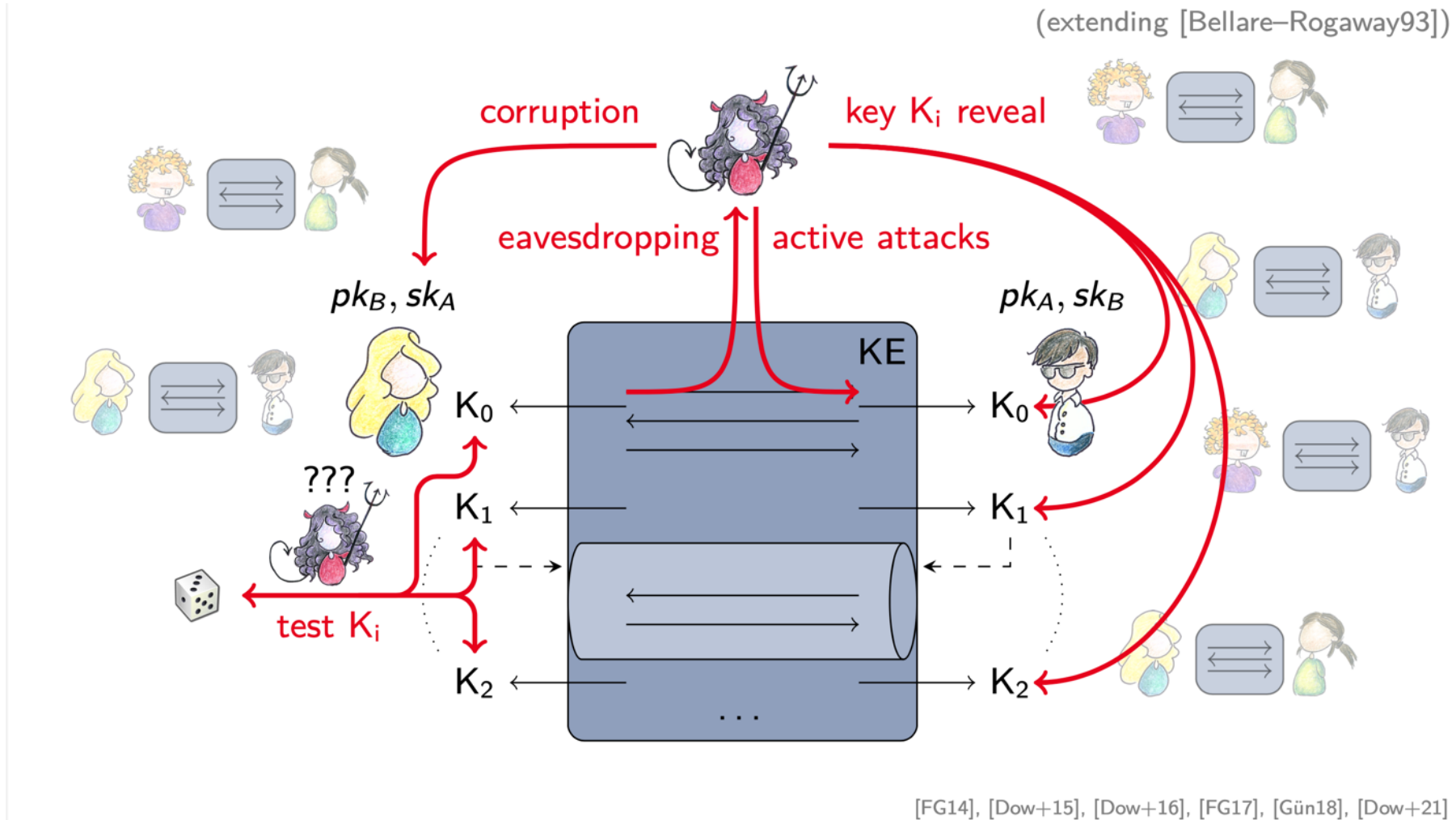
# Multi-stage key exchange security



# Multi-stage key exchange security



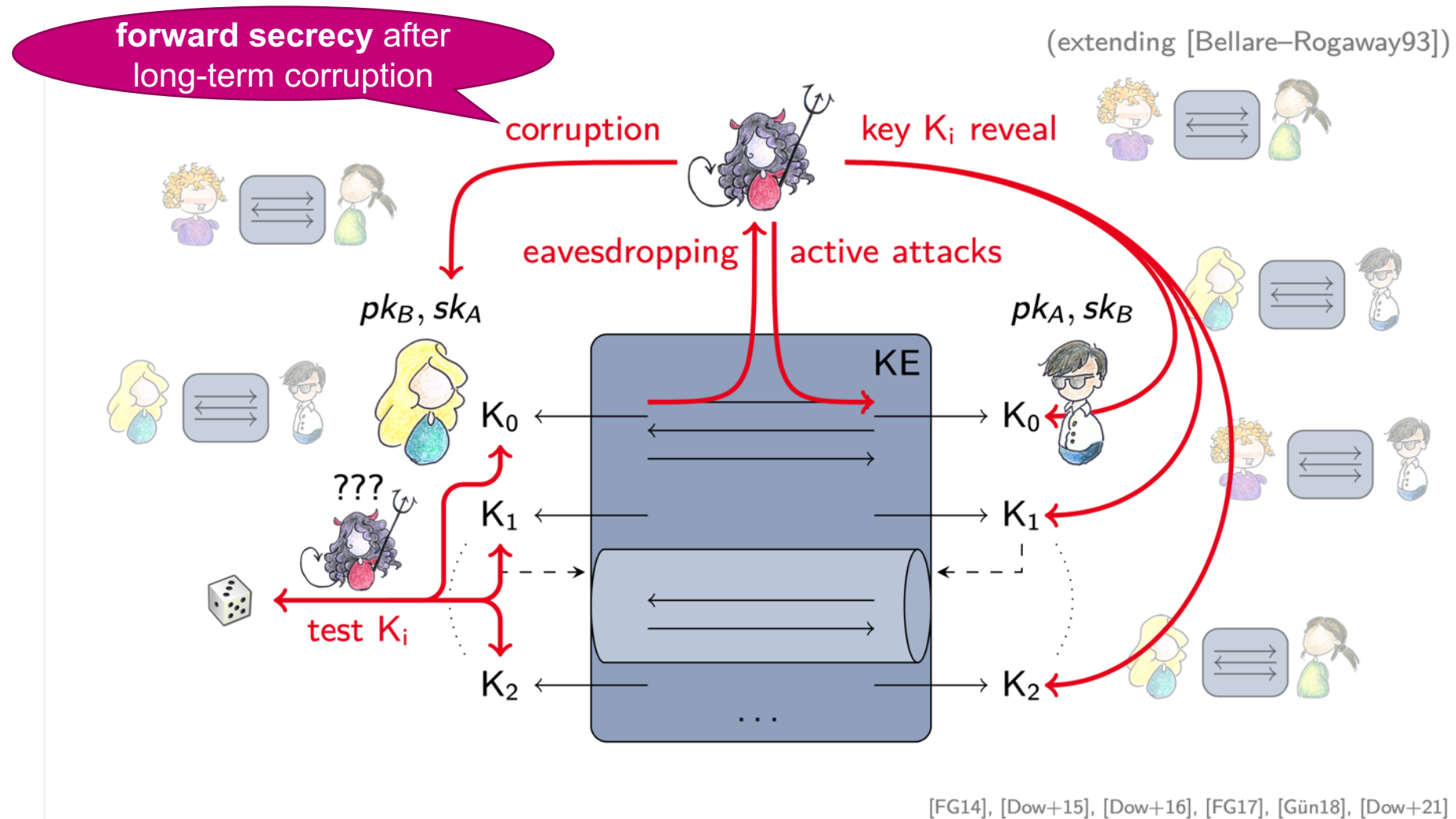
# Multi-stage key exchange security



[FG14], [Dow+15], [Dow+16], [FG17], [Gün18], [Dow+21]



# Multi-stage key exchange security



# 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

# 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)

# 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

Feature	In model of	
	Approach 1	Approach 2
<i>Protocol modelling</i>		
Encrypted handshake messages	✓	✗
HKDF and HMAC decomposed into hash calls	✓	✗
Key exch. and auth. KEMs are the same algorithm	✓	✗
TLS message structure	✓	✗
<i>Security properties</i>		
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

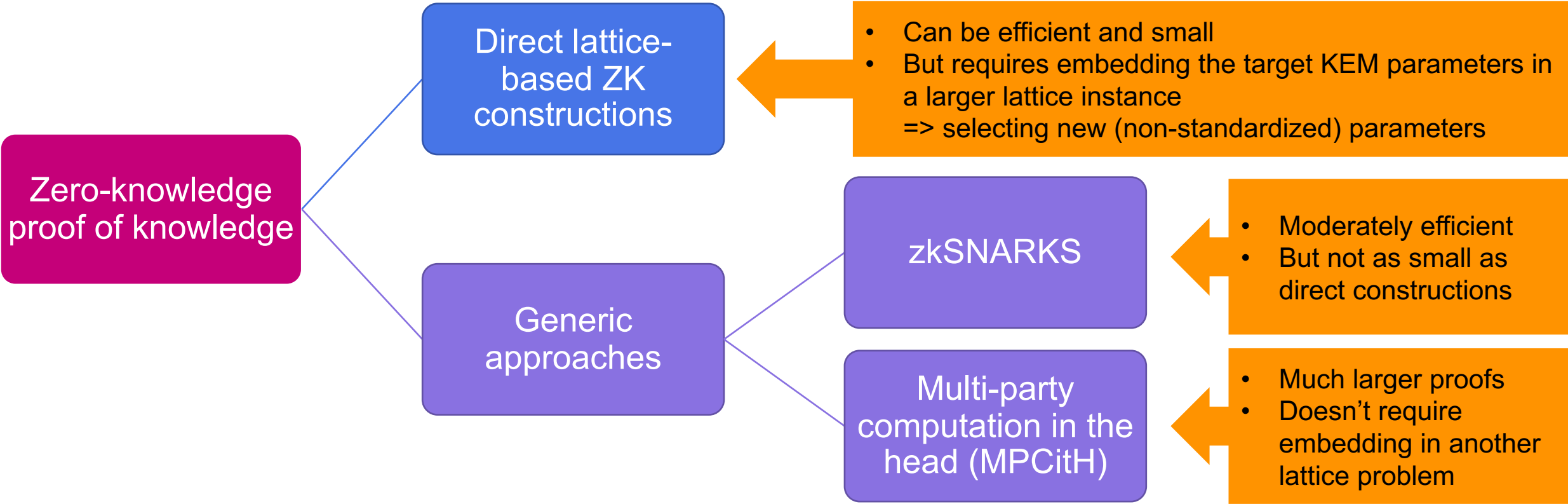
# Goal:

# 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  $\mathbf{S} \leftarrow_{\$} \chi^{n \times \bar{n}}$
3. Sample  $\mathbf{E} \leftarrow_{\$} \chi^{n \times \bar{n}}$
4. Compute  $\mathbf{B} \leftarrow_{\$} \mathbf{AS} + \mathbf{E}$
5. Public key:  $(\text{seed}_{\mathbf{A}}, \mathbf{B})$
6. Secret key:  $\mathbf{S}$

Frodo-640

$$q = 2^{15}$$

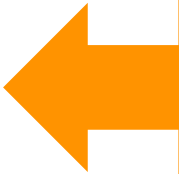
$$n = 640, \bar{n} = 8$$

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




# 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  $\tau$  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

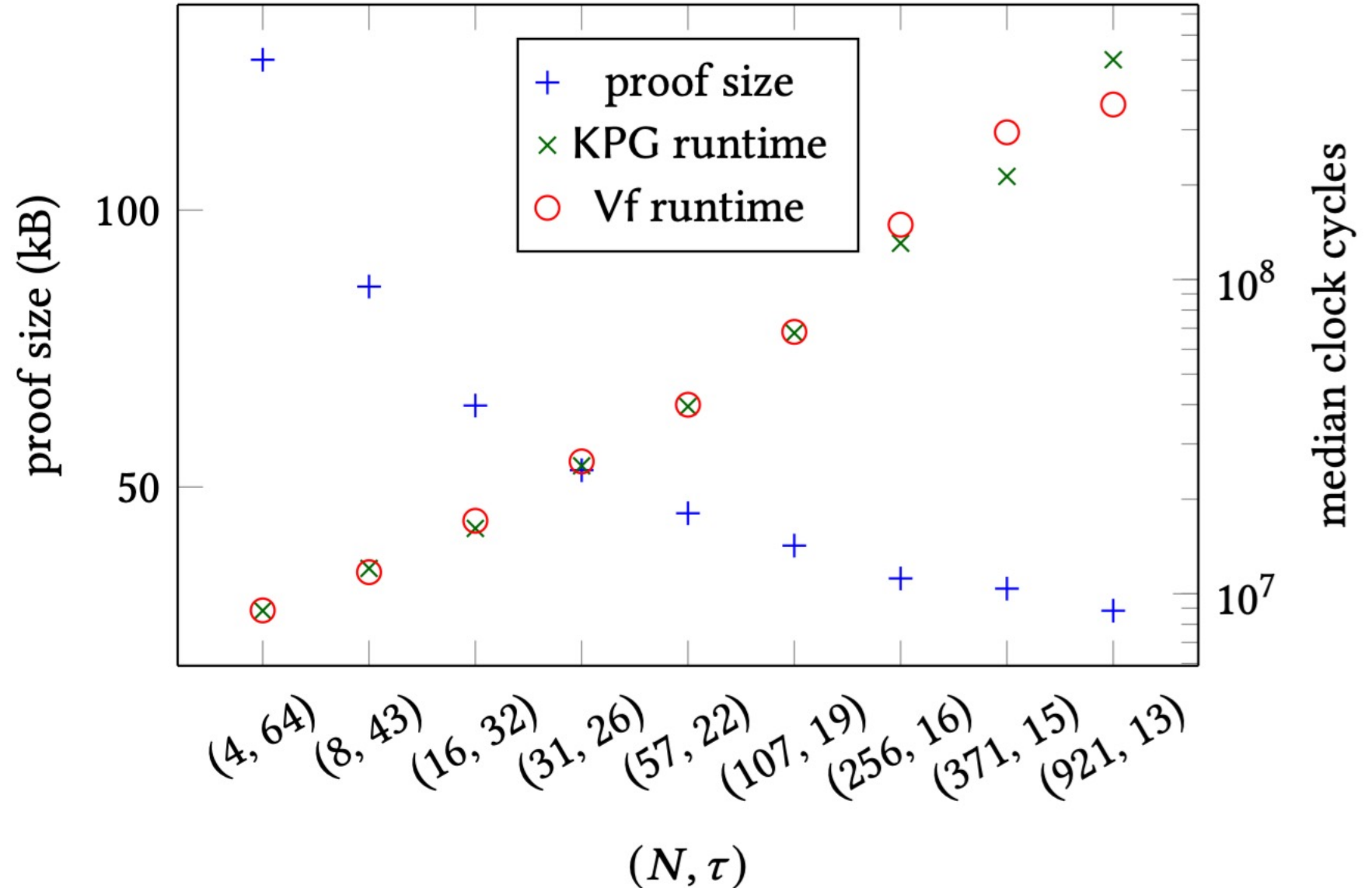
52.9 kB / 0.01s

33.4 kB / 0.03s

25.6 kB / 0.1s

17.8 kB / 3.8s

(a) Kyber-512



# Performance trade-offs

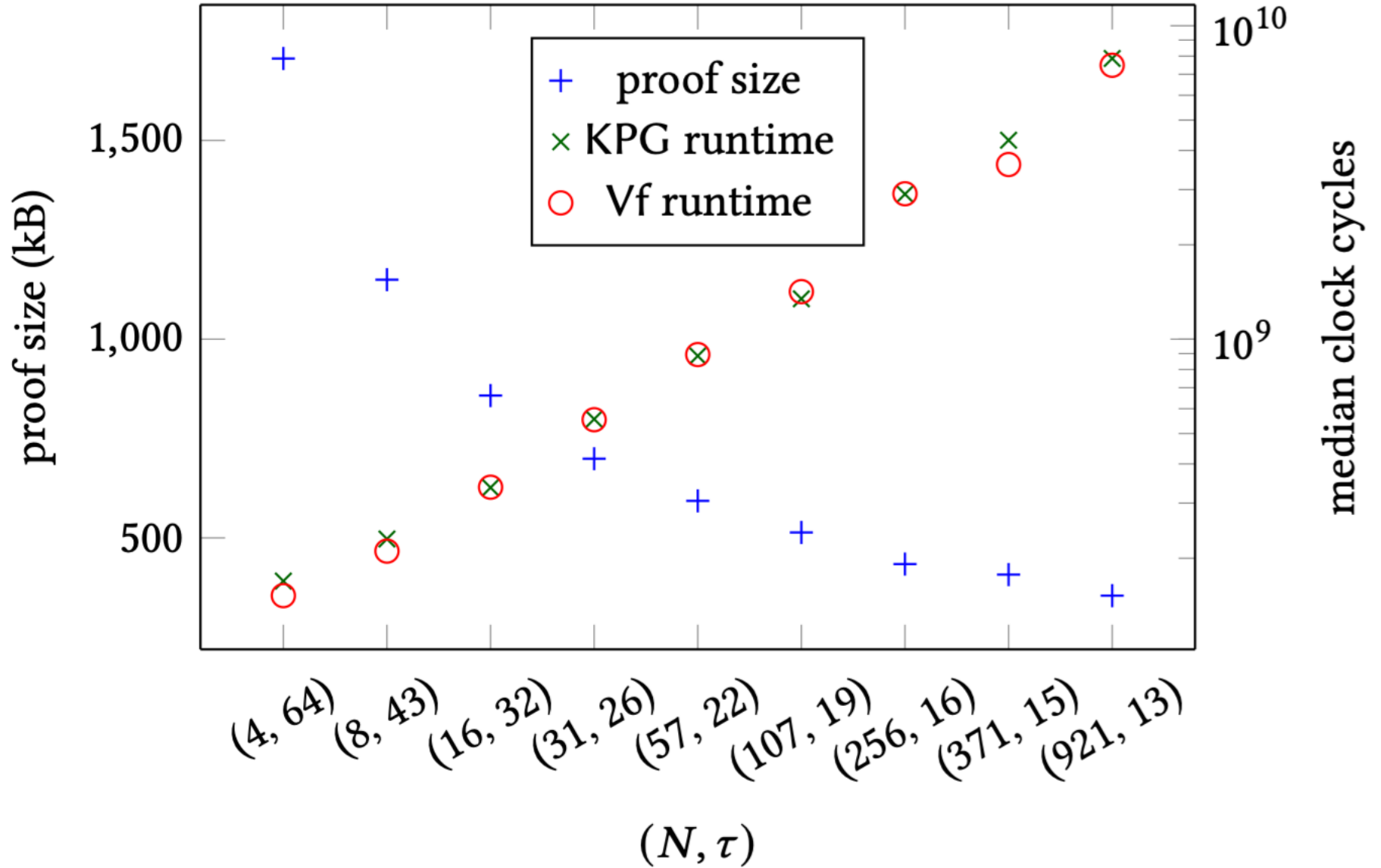
650 kB / 0.1s

402 kB / 0.6s

303 kB / 2.6s

203 kB / 85.6s

(b) FrodoKEM-640





# 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

# Integrating post-quantum cryptography into real-world protocols, part 2

Douglas Stebila



## 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>

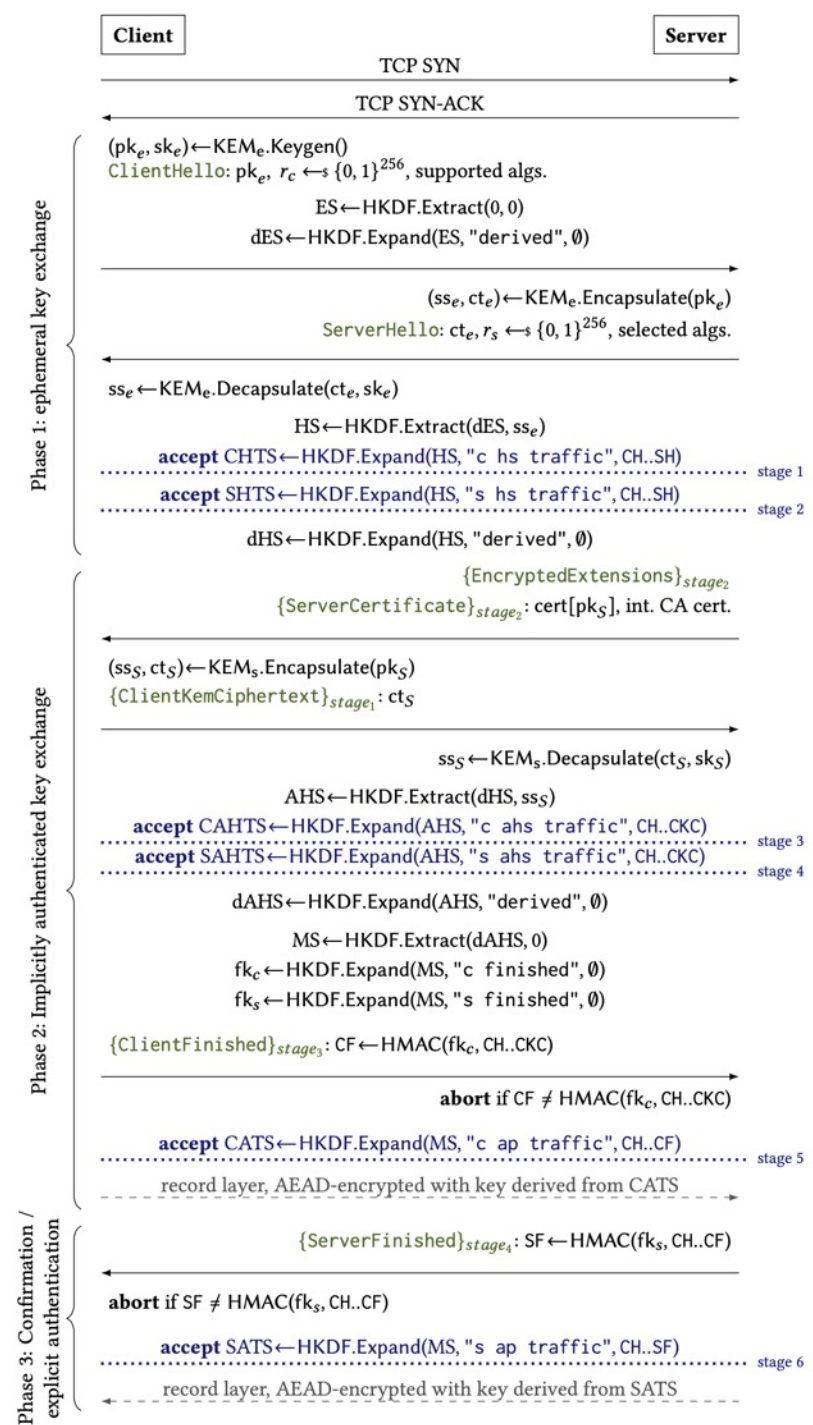
<https://eprint.iacr.org/2020/534> • <https://eprint.iacr.org/2021/779>

<https://datatracker.ietf.org/doc/html/draft-celi-wiggers-tls-authkem-00>

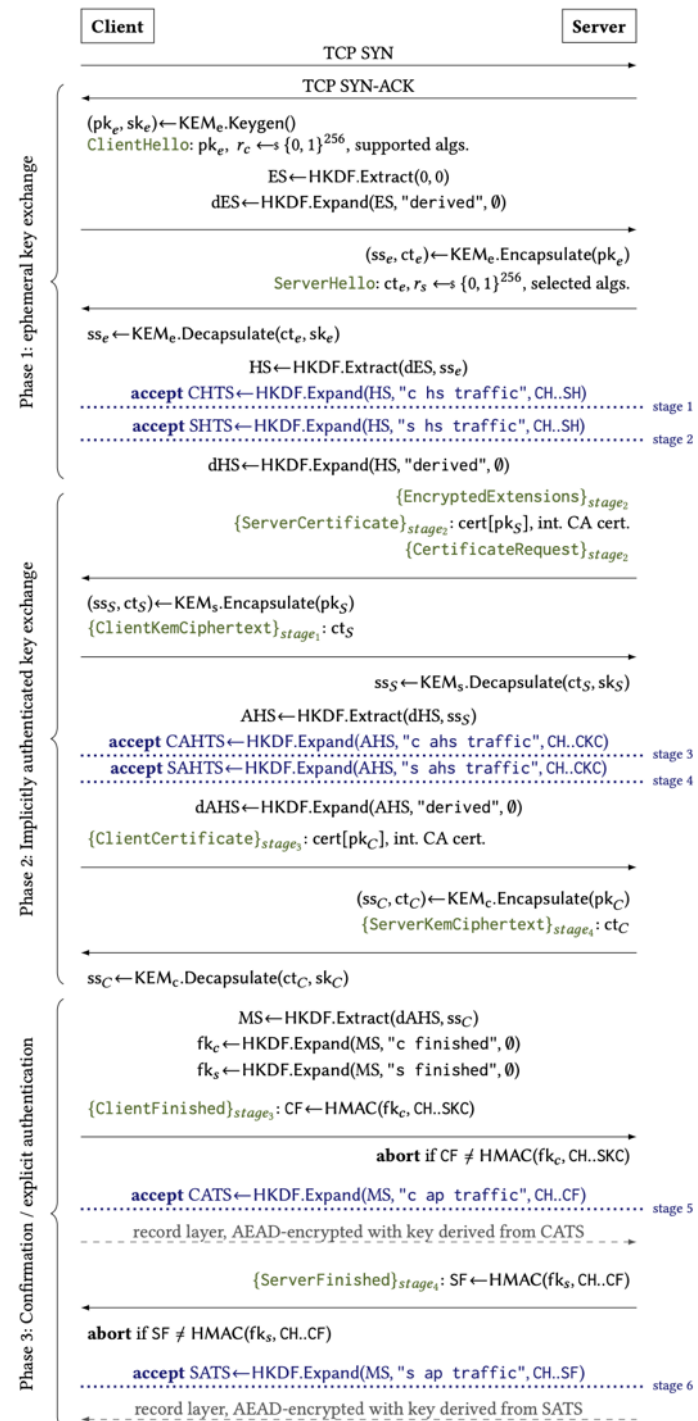
<https://github.com/thomwiggers/TLS13Tamarin> • <https://github.com/dstebila/KEMTLS-Tamarin/>

# Appendix

# KEMTLS



# KEMTLS with client authentication



# TLS 1.3 and KEMTLS size of public key objects

	Abbrev.	KEX (pk+ct)	Excluding intermediate CA certificate					Including intermediate CA certificate			Root CA (pk)	Sum TCP pay-loads of TLS HS (incl. int. CA crt.)
			HS auth (ct/sig)	Leaf crt. subject (pk)	Leaf crt. (signature)	Sum excl. int. CA crt.	Int. CA crt. subject (pk)	Int. CA crt. (signature)	Sum incl. int. CA crt.			
TLS 1.3 (Signed KEX)	<b>TLS 1.3</b>	errr	ECDH (X25519) 64	RSA-2048 256	RSA-2048 272	RSA-2048 256	<b>848</b>	RSA-2048 272	RSA-2048 256	<b>1376</b>	RSA-2048 272	2829
	<b>Min. incl. int. CA cert.</b>	SFXR	SIKE 433	Falcon 690	Falcon 897	XMSS <sub>s</sub> <sup>MT</sup> 979	<b>2999</b>	XMSS <sub>s</sub> <sup>MT</sup> 32	Rainbow 66	<b>3097</b>	Rainbow 161600	5378
	<b>Min. excl. int. CA cert.</b>	SFRR	SIKE 433	Falcon 690	Falcon 897	Rainbow 66	<b>2086</b>	Rainbow 60192	Rainbow 66	<b>62344</b>	Rainbow 60192	64693
	<b>Assumption: MLWE+MSIS</b>	KDDD	Kyber 1568	Dilithium 2420	Dilithium 1312	Dilithium 2420	<b>7720</b>	Dilithium 1312	Dilithium 2420	<b>11452</b>	Dilithium 1312	12639
	<b>Assumption: NTRU</b>	NFFF	NTRU 1398	Falcon 690	Falcon 897	Falcon 690	<b>3675</b>	Falcon 897	Falcon 690	<b>5262</b>	Falcon 897	6524
KEMTLS	<b>Min. incl. int. CA cert.</b>	SSXR	SIKE 433	SIKE 236	SIKE 197	XMSS <sub>s</sub> <sup>MT</sup> 979	<b>1845</b>	XMSS <sub>s</sub> <sup>MT</sup> 32	Rainbow 66	<b>1943</b>	Rainbow 60192	4252
	<b>Min. excl. int. CA cert.</b>	SSRR	SIKE 433	SIKE 236	SIKE 197	Rainbow 66	<b>932</b>	Rainbow 60192	Rainbow 66	<b>61190</b>	Rainbow 60192	63568
	<b>Assumption: MLWE+MSIS</b>	KKDD	Kyber 1568	Kyber 768	Kyber 800	Dilithium 2420	<b>5556</b>	Dilithium 1312	Dilithium 2420	<b>9288</b>	Dilithium 1312	10471
	<b>Assumption: NTRU</b>	NNFF	NTRU 1398	NTRU 699	NTRU 699	Falcon 690	<b>3486</b>	Falcon 897	Falcon 690	<b>5073</b>	Falcon 897	6359

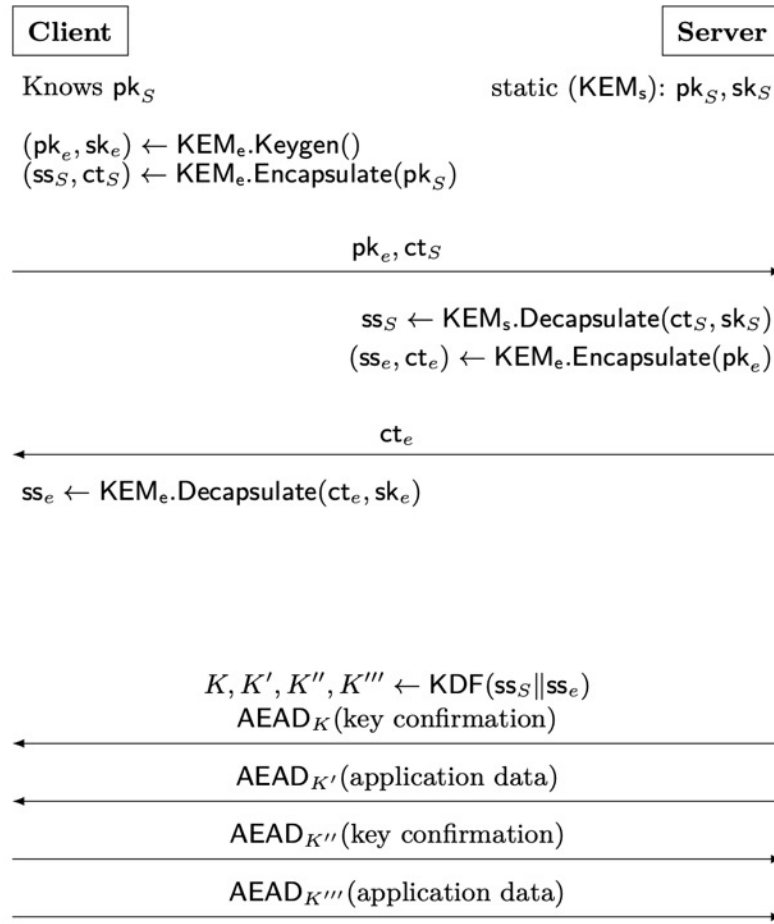
# TLS 1.3 and KEMTLS crypto & handshake time

		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	Client	Server	Client	Client	Server	Client	Client	Server	Client	Client	Server
						sent req.	recv. resp.	HS done	sent req.	recv. resp.	HS done	sent req.	recv. resp.	HS done	sent req.	recv. resp.	HS done
TLS 1.3	<b>errr</b>	0.134	0.629	0.150	0.629	66.4	<b>97.7</b>	35.5	66.5	<b>97.7</b>	35.5	397.3	<b>593.4</b>	201.4	398.3	<b>594.5</b>	202.4
	<b>SFXR</b>	11.860	4.410	12.051	4.410	80.1	<b>111.3</b>	49.2	80.4	<b>111.5</b>	49.4	417.5	<b>615.0</b>	218.9	417.4	<b>614.9</b>	219.1
	<b>SFRR</b>	6.061	4.410	6.251	4.410	65.5	<b>96.7</b>	34.5	131.4	<b>162.6</b>	100.4	398.3	<b>594.6</b>	201.8	1846.8	<b>2244.5</b>	1578.7
	<b>KDDD</b>	0.059	0.072	0.081	0.072	63.8	<b>95.1</b>	32.9	64.1	<b>95.4</b>	33.2	405.1	<b>602.3</b>	208.3	410.3	<b>609.8</b>	212.8
	<b>NFFF</b>	0.138	0.241	0.180	0.241	64.8	<b>96.0</b>	33.8	65.1	<b>96.4</b>	34.2	397.8	<b>593.9</b>	201.2	399.8	<b>596.0</b>	203.2
KEMTLS	<b>SSXR</b>	15.998	7.173	16.188	7.173	84.5	<b>124.6</b>	62.5	84.3	<b>124.4</b>	62.3	417.5	<b>625.8</b>	232.5	417.6	<b>625.8</b>	232.4
	<b>SSRR</b>	10.198	7.173	10.388	7.173	75.5	<b>116.3</b>	54.2	140.3	<b>182.3</b>	120.1	408.5	<b>616.5</b>	223.5	1684.2	<b>2091.6</b>	1280.4
	<b>KKDD</b>	0.048	0.017	0.070	0.017	63.3	<b>94.8</b>	32.6	63.7	<b>95.2</b>	32.9	397.3	<b>594.4</b>	201.6	434.7	<b>638.0</b>	235.4
	<b>NNFF</b>	0.107	0.021	0.149	0.021	63.4	<b>95.0</b>	32.7	63.7	<b>95.3</b>	33.0	395.9	<b>593.0</b>	200.1	397.6	<b>594.7</b>	201.9

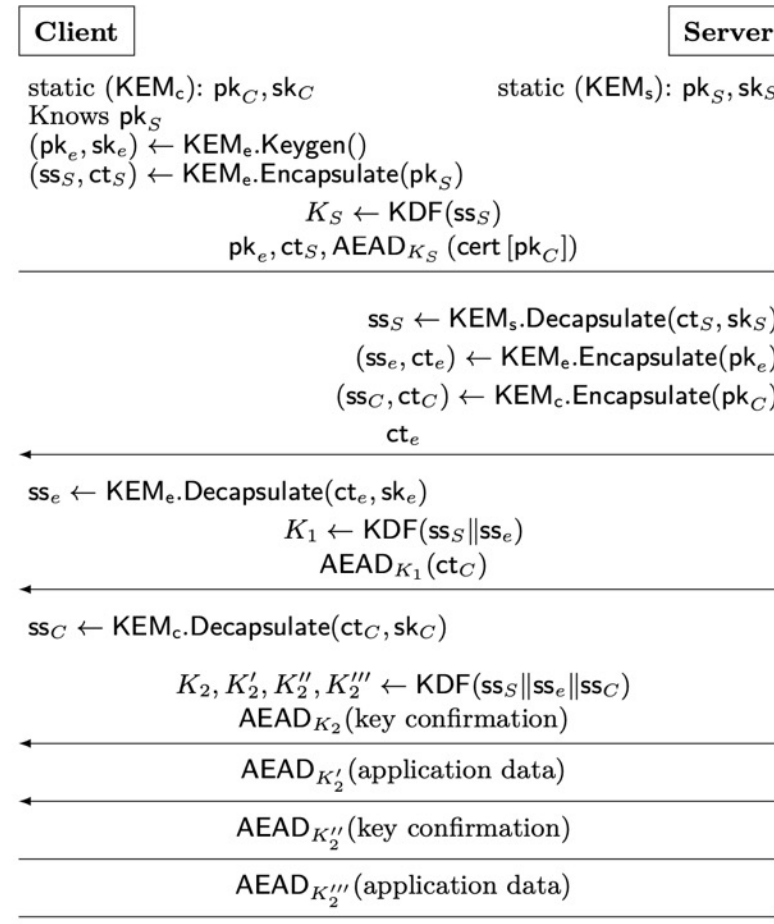
Label syntax: ABCD: A = ephemeral key exchange, B = leaf certificate, C = intermediate CA certificate, D = root certificate.

Label values: Dilithium, eCDH X25519, Falcon, Kyber, NTRU, Rainbow, rSA-2048, SIKE, XMSS<sub>s</sub><sup>MT</sup>; all level-1 schemes.

# KEMTLS-PDK overview



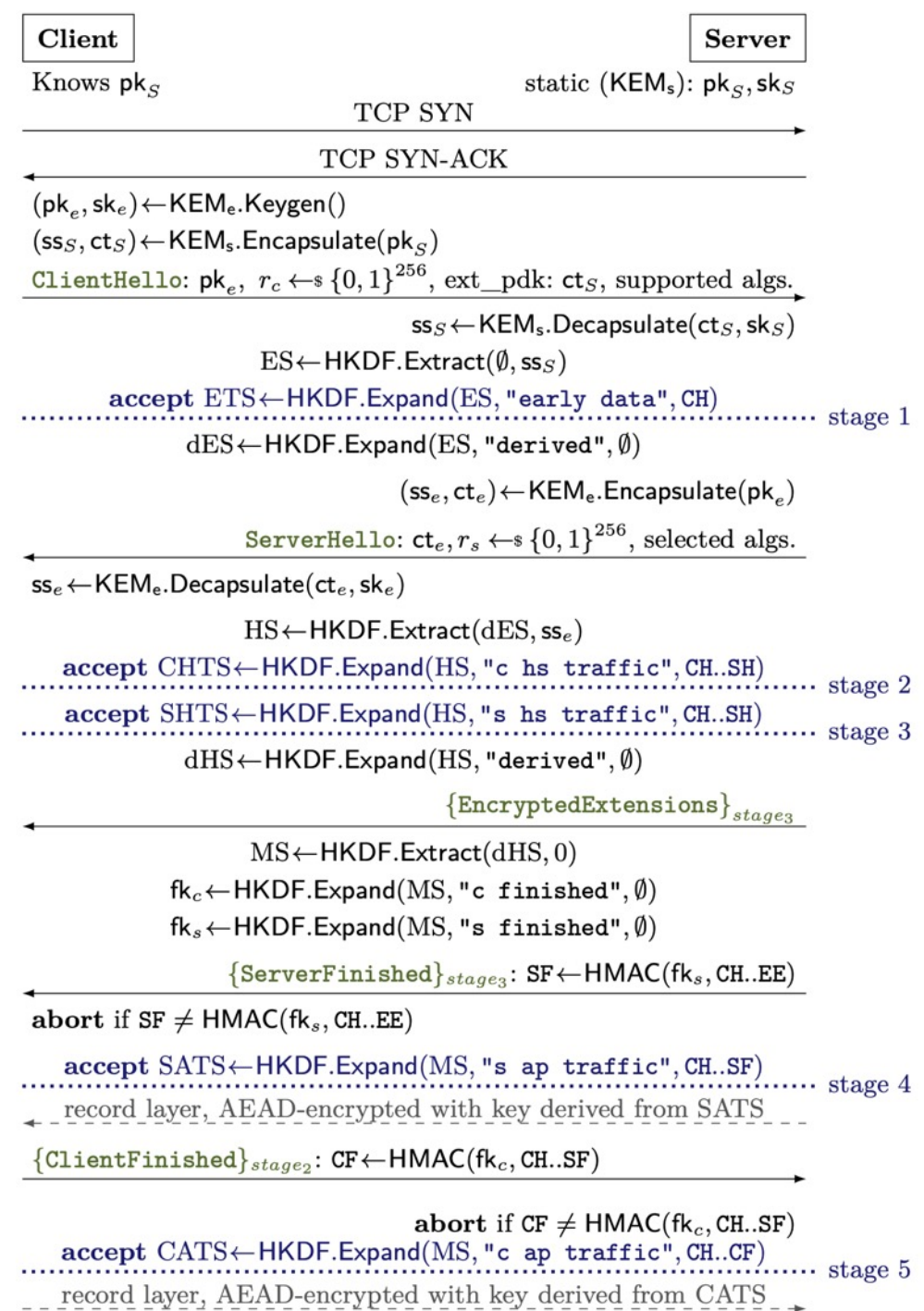
(a) Unilaterally authenticated



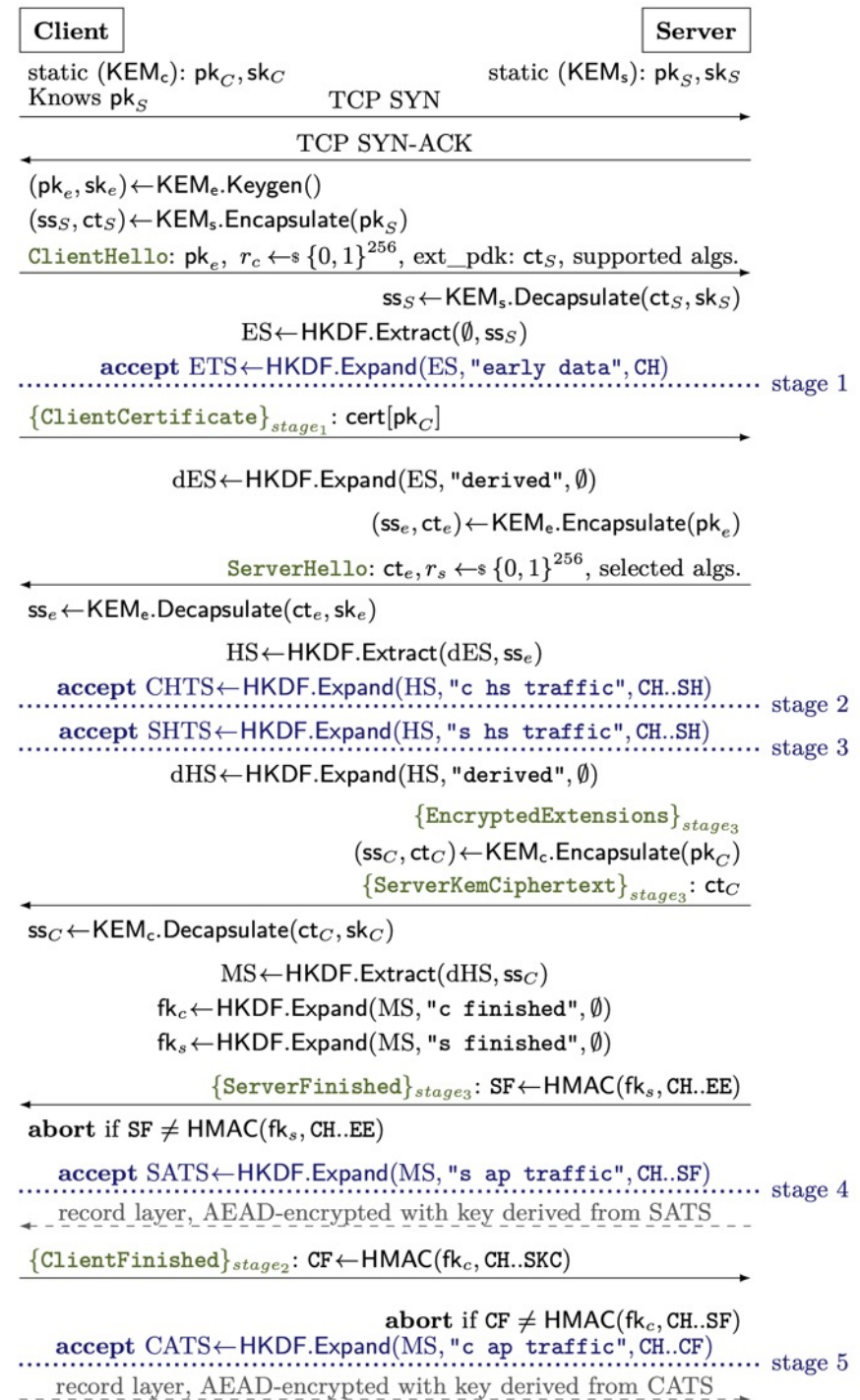
(b) With proactive client authentication



# KEMTLS-PDK



# KEMTLS-PDK with proactive client authentication



# Communication sizes

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

KEMTLS

TLS 1.3 w/cached server certs

KEMTLS-PDK

# Handshake times, unilateral authentication

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

# Handshake times, mutual authentication

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

	KEMTLS	Cached TLS	KEMTLS-PDK
<i>Unilaterally authenticated</i>			
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
<i>Mutually authenticated</i>			
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 variants
	sauth	mutual	both	sauth	mutual	both	
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
<i>Proof of knowledge of secret key (and proof of verifiable decryption, denoted <math>\diamond</math>)</i>								
Stern-like [51]	ZKP from SIS	2.3 MB <sup>†</sup>	4.3 MB <sup>†</sup>					
[9]	MPCitH		4.1 MB	2.4 s			$\geq 8.42$ MB <sup>‡</sup>	
[20]	ZKP from RLWE & RSIS	384 kB <sup>†</sup>						
[13]	$\Sigma$ -prot. for permuted-kernel	233 kB	444 kB					
Ligero [5]	zkSNARK from PCPs	157 kB <sup>†</sup>	200 kB <sup>†</sup>					
Aurora [11]	zkSNARK for R1CS	72 kB <sup>†</sup>	71 kB <sup>†</sup>					
[37]	ZKP from MLWE & MSIS	47 kB, 61 kB $\diamond$						
[56]	ZKP from MLWE & MSIS	47 kB*						
[57]	ZKP from MSIS & ext. MLWE	33 kB			43.6 kB $\diamond$			
[55]	ZKP from MLWE & MSIS	14 kB			19.0 kB $\diamond$			
<i>Proof of verifiable generation</i>								
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 ( $N, \tau$ )” denotes number  $N$  of MPC-in-the-head parties and number  $\tau$  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.

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

\*: Estimate by Lyubashevsky et al. [57].  $\diamond$ : 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.