Looma

A Low-latency PQ Mutual Authentication Architecture

draft-ma-cfrg-looma-00

Xinshu Ma, Michio Honda (University of Edinburgh),
Colin Perkins (University of Glasgow)

Context & Goal

- Looma: a research prototype for low-latency PQ authentication in TLS 1.3
  - We are exploring whether this could be useful for real-world deployment
  - Not proposing standardisation at this stage

- Before considering taking this to TLS, we would like feedback from CFRG

- Goal of this talk:
  - Validate that our construction and composition are sound

Why Fast Authentication Matters

Cloud needs fast mutual authentication

- Many latency-sensitive short flows
  - Crypto overhead can be dominant

- Sub-ms flow RTT over high bandwidth fabric
  - Crypto overhead is compute-bound



Software techniques that tolerate latency variability are vital to building responsive large-scale Web services.

BY JEFFREY DEAN AND LUIZ ANDRÉ BARROSO

The Tail at Scale

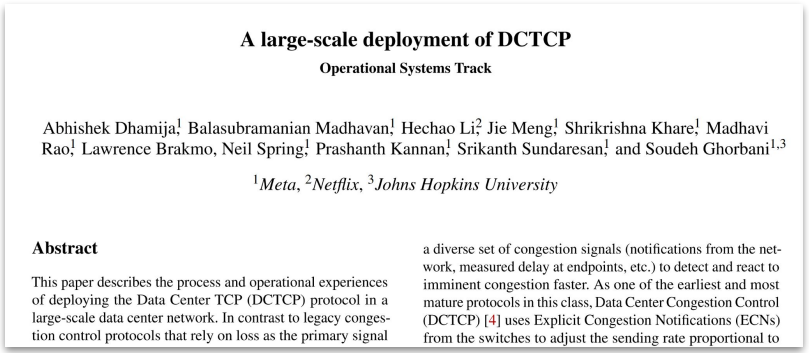
SYSTEMS THAT RESPOND to user actions quickly (within 100ms) feel more fluid and natural to users than

high-latency episodes (unimportant in moderate-size systems) may come to dominate overall service performance at large scale. Just as fault-tolerant computing aims to create a reliable whole out of less-reliable parts, large online services need to create a predictably responsive whole out of less-predictable parts; we refer to such systems as "latency tail-tolerant," or simply "tail-tolerant." Here, we outline some common causes for high-latency episodes in large online services and describe techniques that reduce their severity or mitigate their effect on whole-system performance. In many cases, tail-tolerant techniques can take advantage of resources already deployed to achieve fault-tolerance, resulting in low additional overhead. We explore how these techniques allow system utilization to be driven higher without lengthening the latency tail, thus avoiding wasteful overprovisioning.

Why Variability Exists?

Variability of response time that leads to high tail latency in individual components of a service can arise for many reasons, including:

Shared resources. Machines might be shared by different applications



A large-scale deployment of DCTCP

Operational Systems Track

Abhishek Dhamija,[1] Balasubramanian Madhavan,[1] Hechao Li,[2] Jie Meng,[1] Shrikrishna Khare,[1] Madhavi Rao,[1] Lawrence Brakmo, Neil Spring,[1] Prashanth Kannan,[1] Srikanth Sundaresan,[1] and Soudeh Ghorbani[1,3]

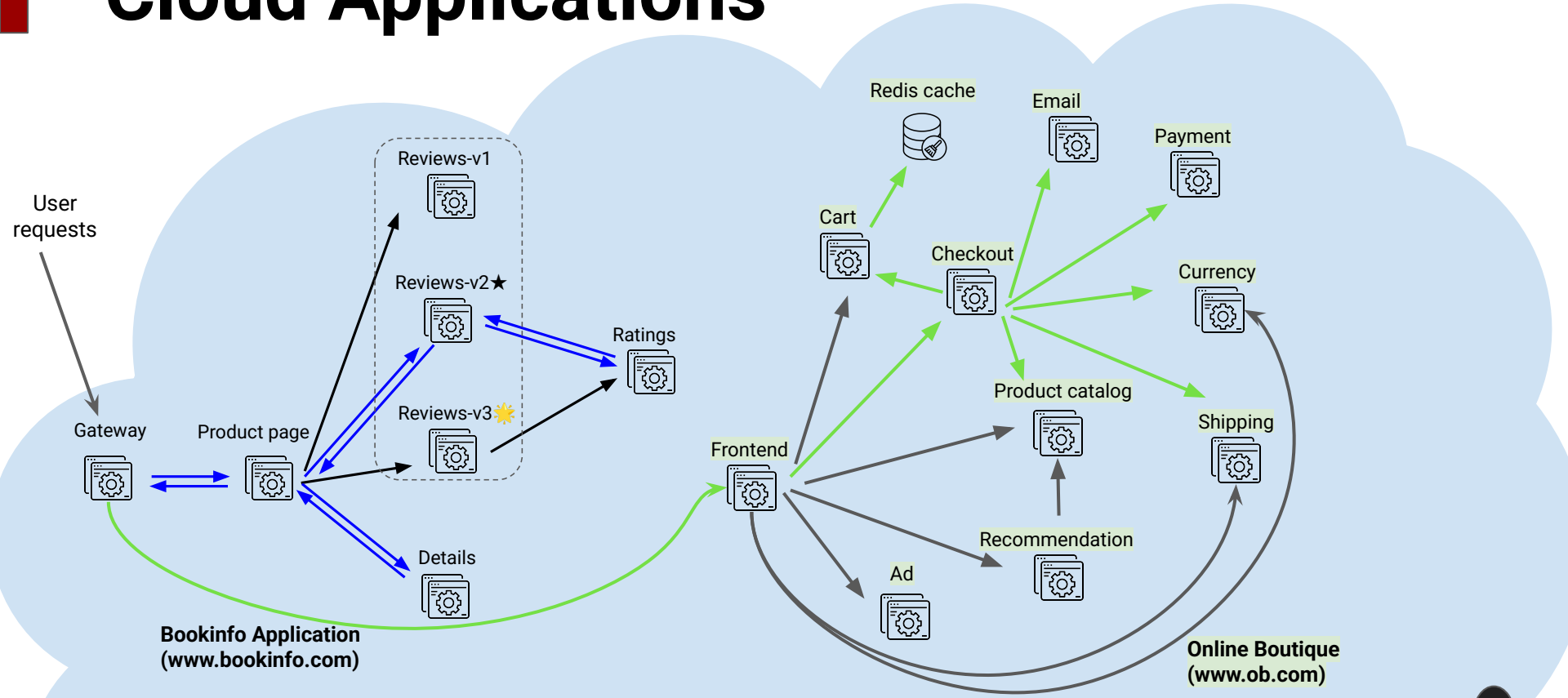
[1]*Meta*, [2]*Netflix*, [3]*Johns Hopkins University*

Abstract

This paper describes the process and operational experiences of deploying the Data Center TCP (DCTCP) protocol in a large-scale data center network. In contrast to legacy congestion control protocols that rely on loss as the primary signal

a diverse set of congestion signals (notifications from the network, measured delay at endpoints, etc.) to detect and react to imminent congestion faster. As one of the earliest and most mature protocols in this class, Data Center Congestion Control (DCTCP) [4] uses Explicit Congestion Notifications (ECNs) from the switches to adjust the sending rate proportional to

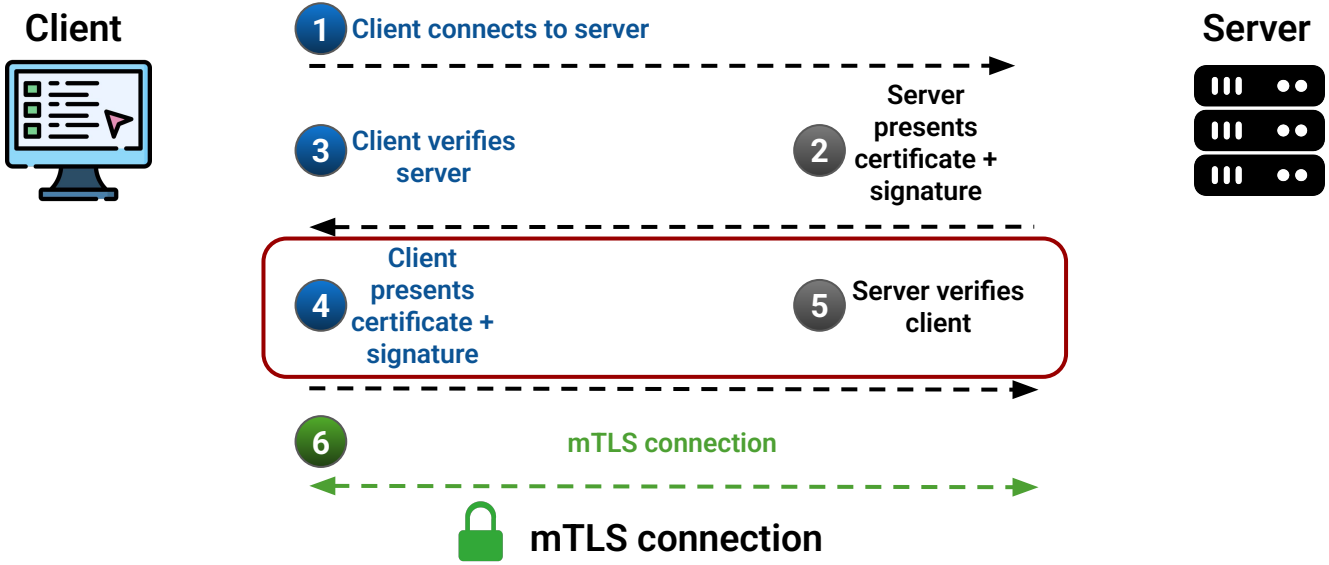
Cloud Applications



https://github.com/GoogleCloudPlatform/microservices-demo

Mutual Authentication

Example: mTLS



Handshake	Keygen+ex	Verify cert	Sign	Verify
$sTLS_c$	●	●	○	●
$sTLS_s$	●	○	●	○
$mTLS_{c,s}$	●	●	●	●

PQ Authentication Amplies the Pain

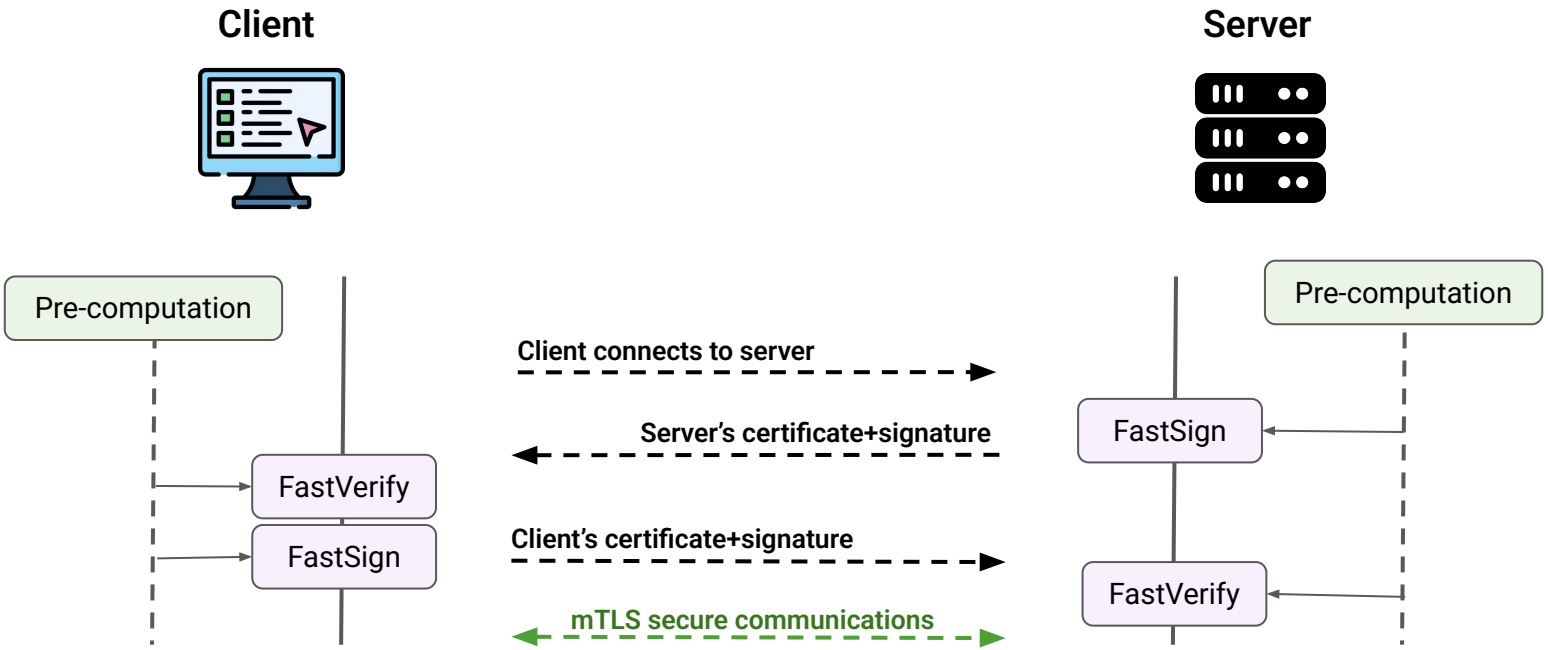
Example: Handshake crypto overhead breakdown

Operation	Algorithm	Latency (µs)	PQ	Library
Keygen+ex	ECDHE[a]	45.20	○	OpenSSL
	KEM[b]	74.70	●	OQS
Verify cert	RSA-2048	17.22	○	OpenSSL
	Dilithium-2	26.27	●	OQS
	Falcon-512	30.7	●	OQS
Sign	ECDSA	14.92	○	OpenSSL
	RSA-2048	200.37	○	OpenSSL
PQ	Dilithium-2	51.74	●	OQS
PQ	Falcon-512	149.2	●	OQS
Verify	ECDSA	40.18	○	OpenSSL
	RSA-2048	15.32	○	OpenSSL
PQ	Dilithium-2	21.27	●	OQS
PQ	Falcon-512	28.3	●	OQS

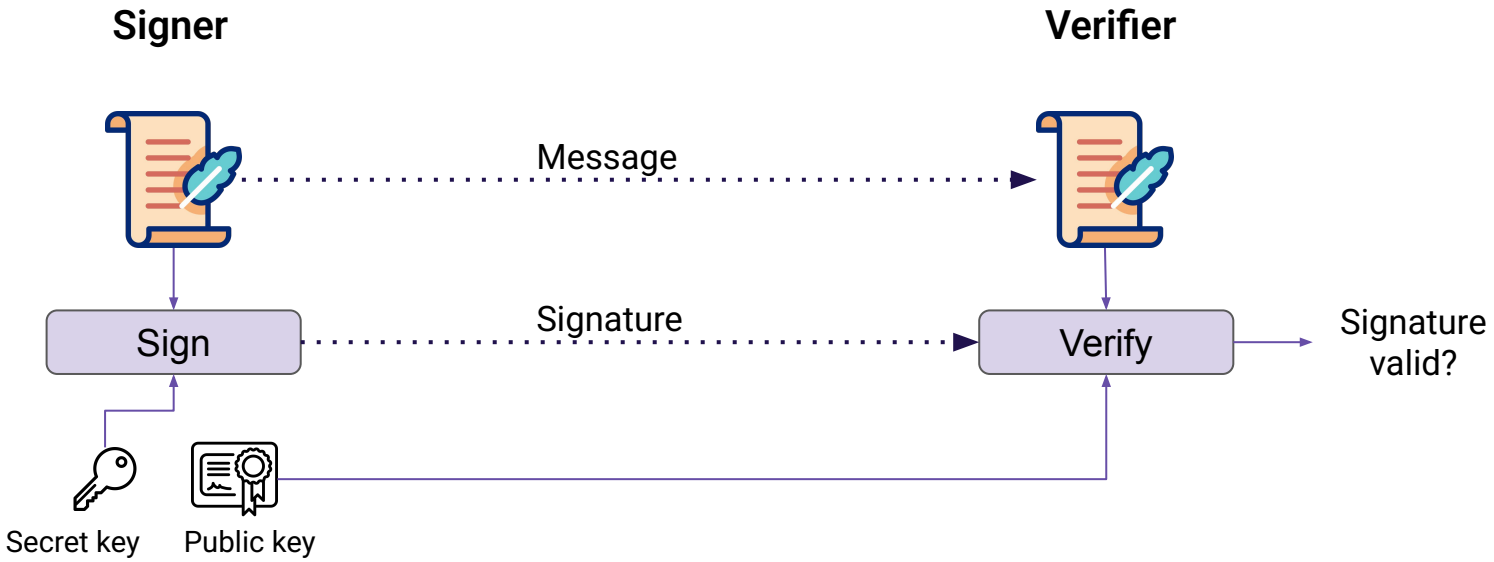
PQ primitives have cheap verify but expensive sign.

Looma: Decoupling Expensive Auth

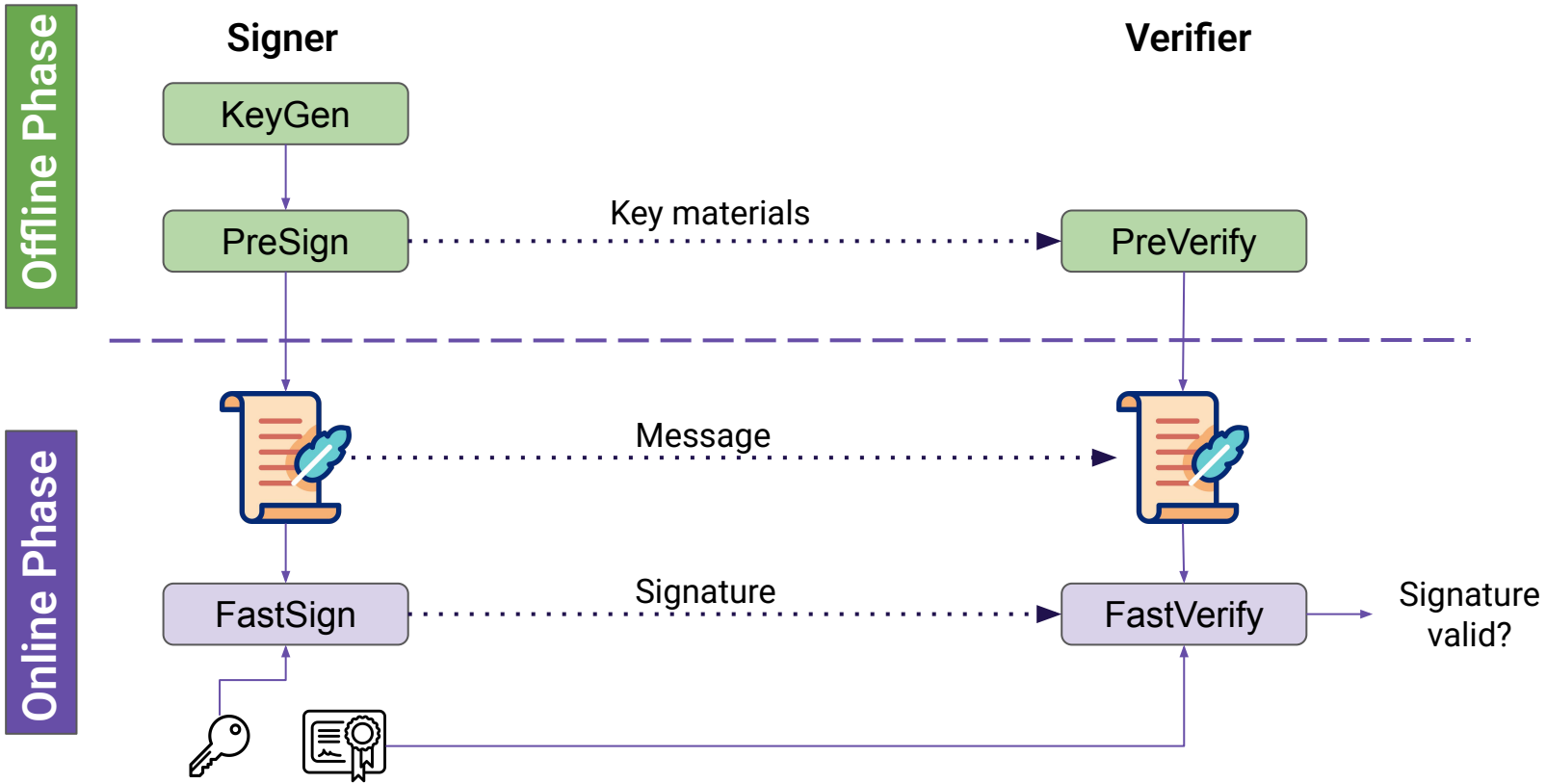
Do expensive work early; keep the online path fast



Normal Signature Workflow



Online/Offline Signature Paradigm



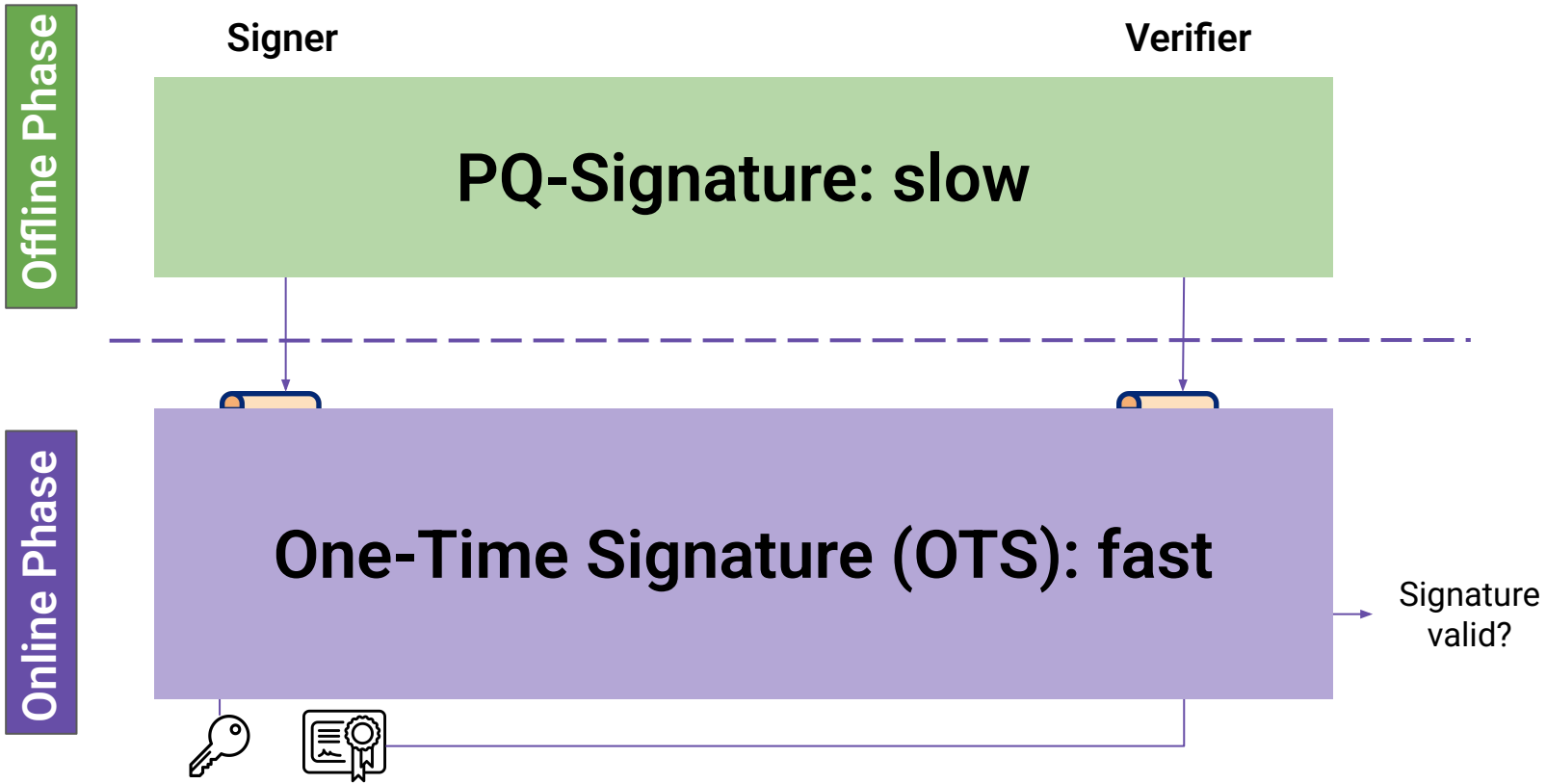
PQ-sig ⟶ Online/Offline PQ-sig

Online/offline signature construction

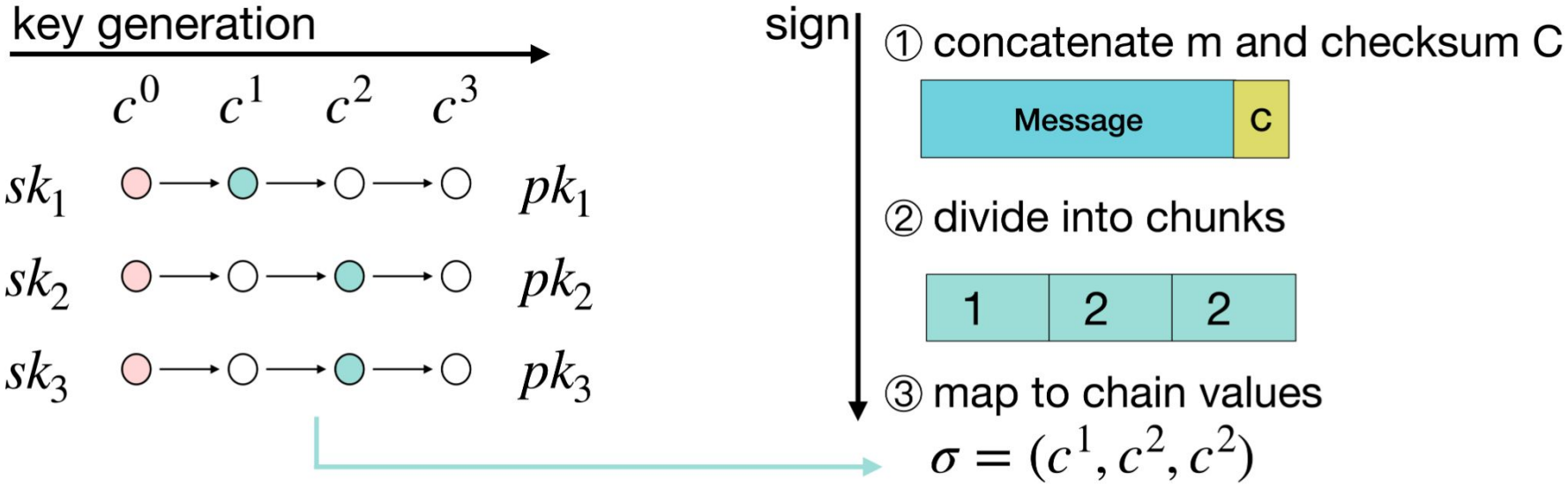
- **Even, Goldreich, Micali [CRYPTO ’89]**
  - **Idea:** one-time signature online
  - **Pros:**
    - simple and modular;
    - no extra primitive
  - **Cons:** larger signature size

- **Shamir and Tauman [CRYPTO ‘01]**
  - **Idea:** trapdoor/chameleon hash online
  - **Pros:** more compact signatures
  - **Cons:** needs an additional PQ chameleon-hash primitive

EGM Online/Offline Signature Paradigm

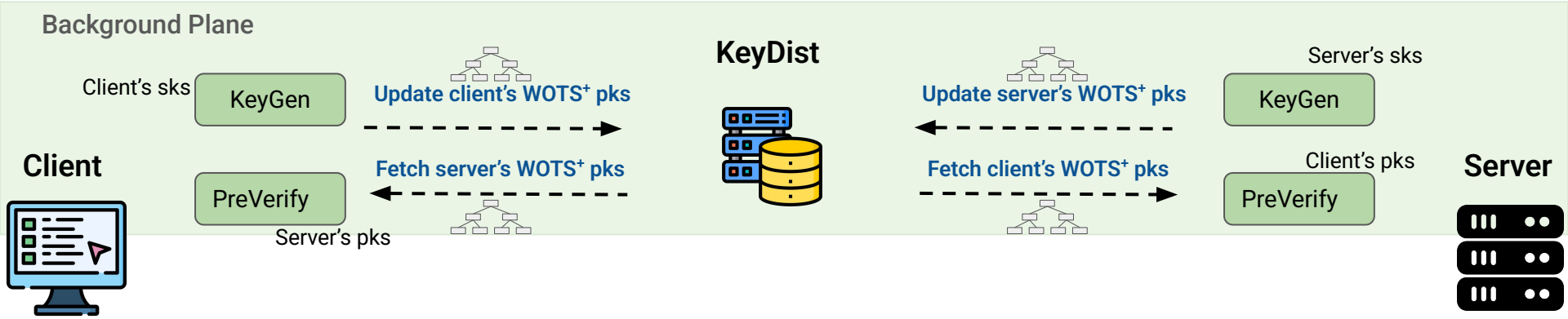


Winternitz One-Time Signature Plus

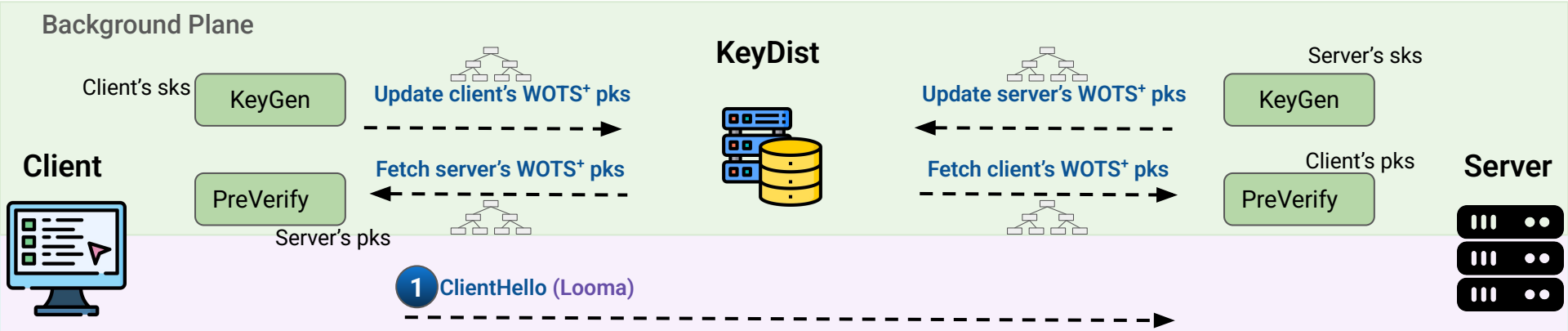


Both SPHINCS+ and XMSS [RFC8391] adopt WOTS$^+$ as leaf signing.

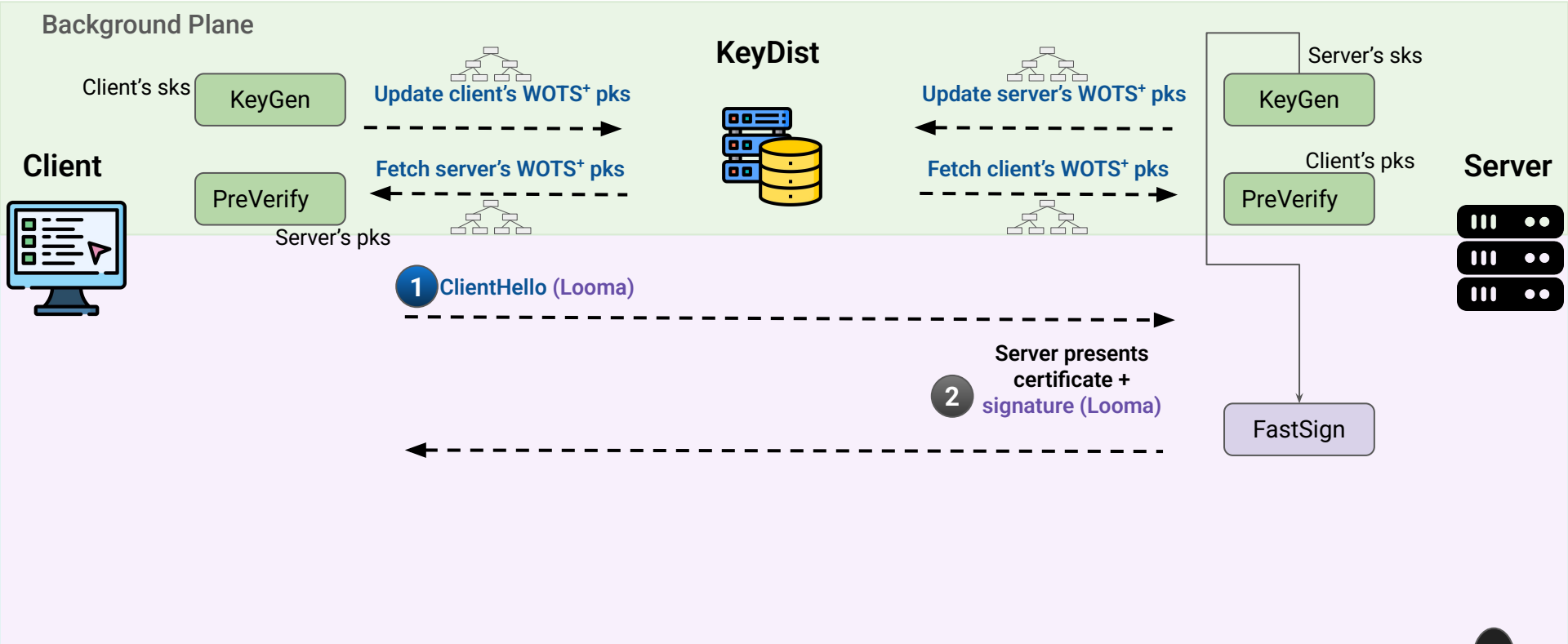
Use Case: KeyDist + Offline Tasks



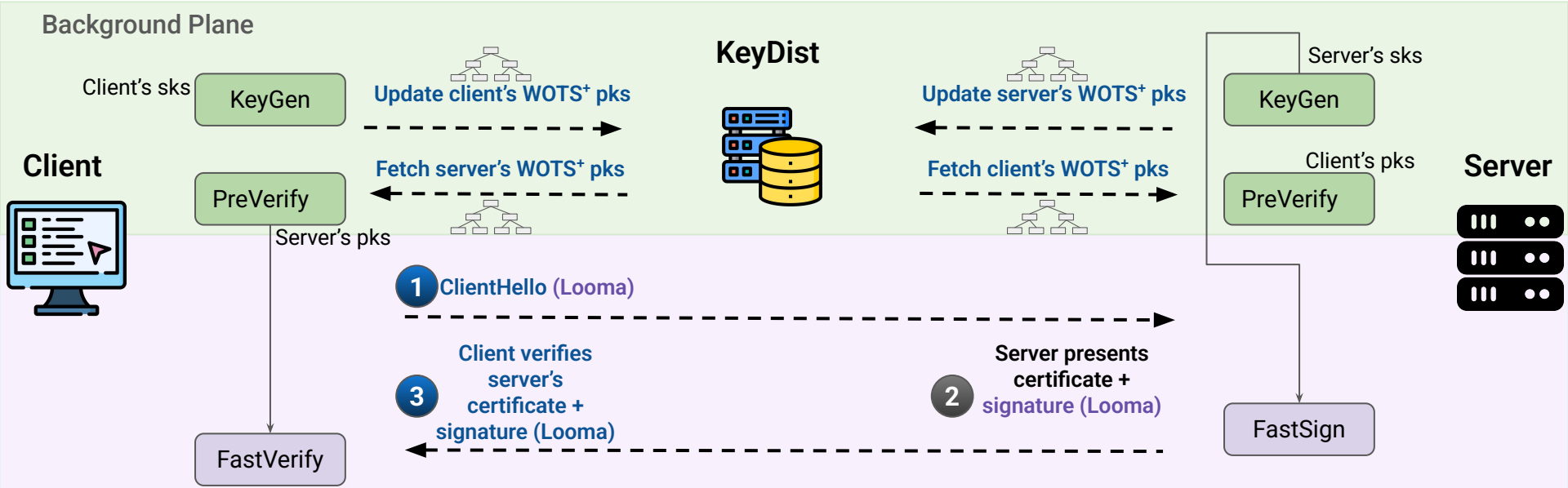
Use case: PQ mTLS Handshake



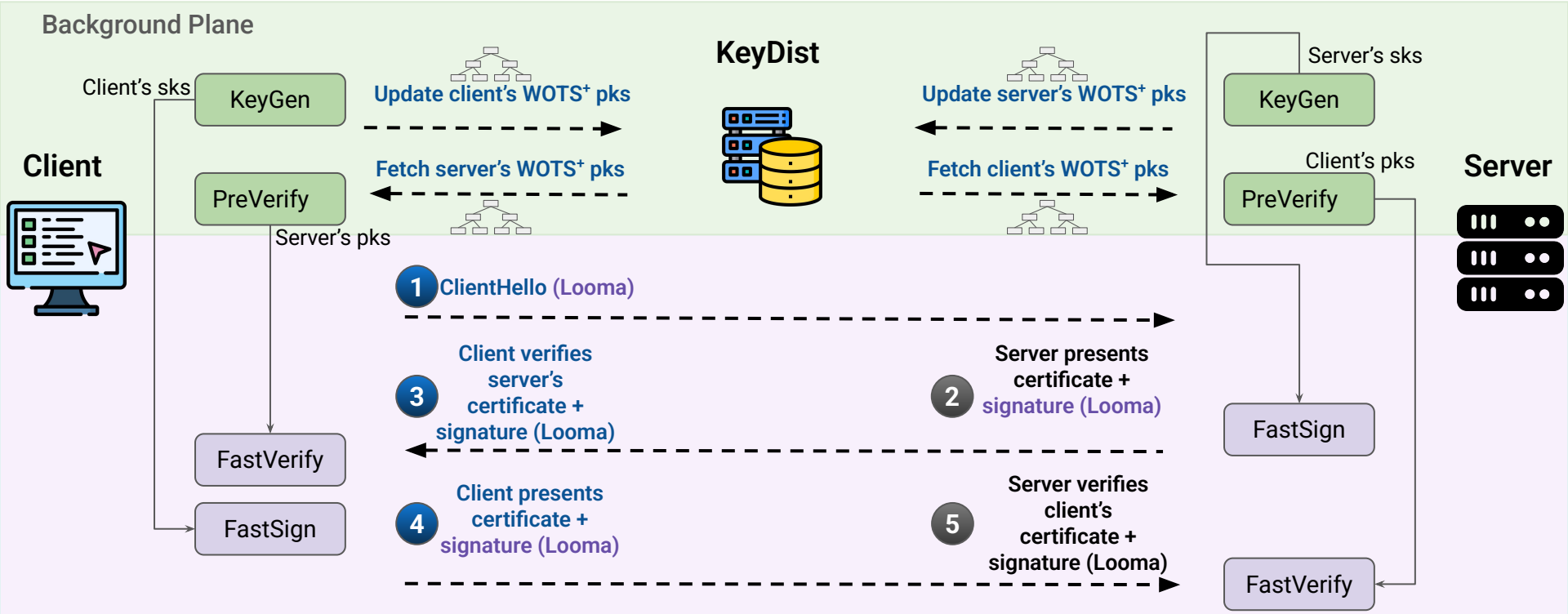
Use case: PQ mTLS Handshake



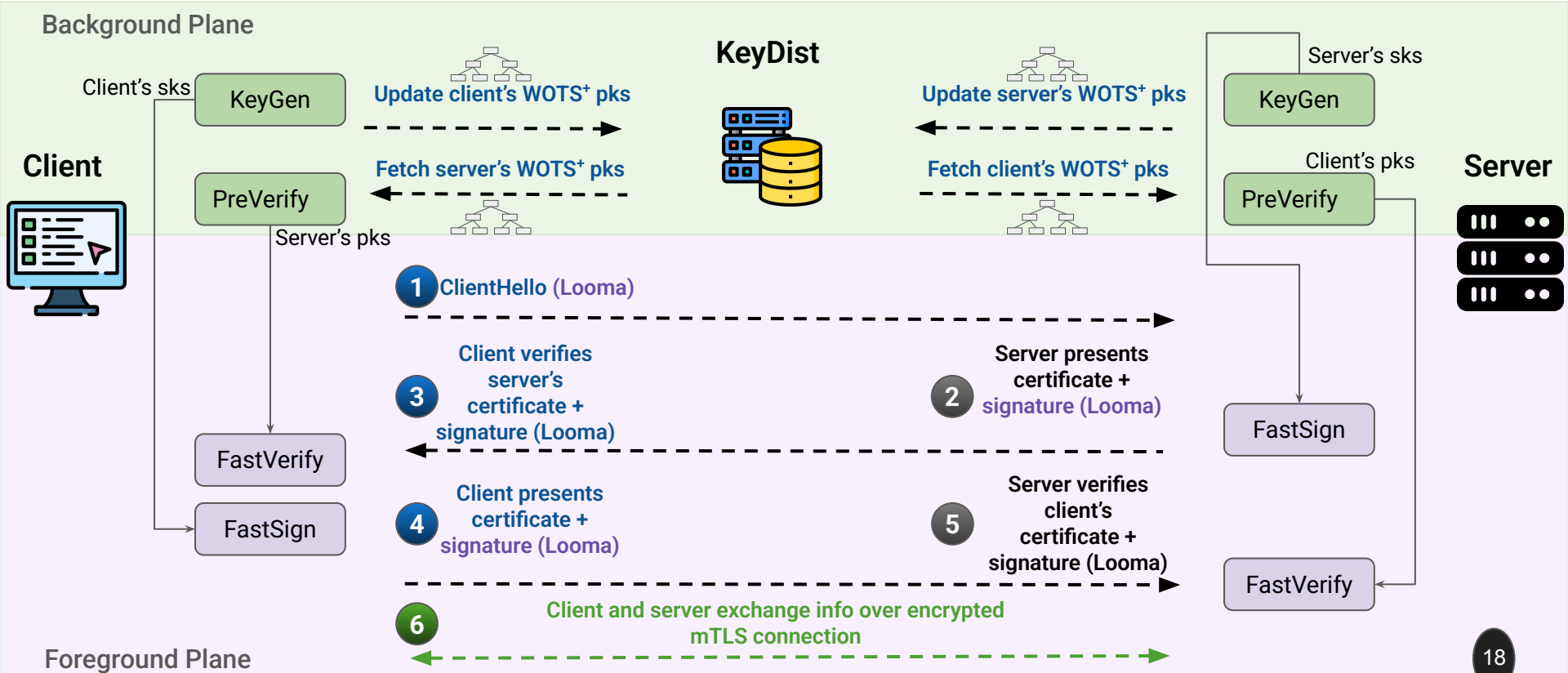
Use case: PQ mTLS Handshake



Use case: PQ mTLS Handshake



Use case: PQ mTLS Handshake



Looma Microbenchmarks: (Sign/Verify)

TABLE VII: Sign and verification latency (in μs).

	I2		A2	
Algorithm	Sign	Verify	Sign	Verify
RSA-2048	261.1	19.3	200.7	15.4
ECDSA-256	21.1	62.6	14.9	40.8
Ed25519	27.6	85.2	18.7	54.4
Dilithium-2	61.4	24.4	51.4	21.3
Falcon-512	181.1	34.3	149.2	28.3
SPHINCS$^+$-SHA2-128f	6866.2	524.9	5611.6	426.3
Looma	**0.39**	**6.83**	**0.32**	**6.19**

Looma Performance (One-to-One)

TABLE VI: TLS handshake latency summary (µs).

Signature Algorithm	sTLS				mTLS			
	Mean	St. Dev.	P50	P99	Mean	St. Dev.	P50	P99
RSA-2048	842.4	206.3	900	1,427	1,095.4	179.2	1,112	1,656
ECDSA-256	422.0	38.9	416	513	462.6	38.6	449	555
ED25519	417.6	47.4	411	511	429.3	57.0	432	534
Dilithium-2	499.9	106.2	485	837	560.5	112.4	548	902
Falcon-512	778.9	117.6	801	902	969.9	56.6	963	1,066
SPHINCS$^+$ SHA256-128f-simple	13,358.5	3,611.7	12,725	18,894	35,289.7	1,264.3	35,471	36,099
Looma	**354.9**	**42.3**	**348**	**447**	**363.6**	**46.2**	**363**	**464**

34% **48%**

Not only improves the median, but also cuts the tail significantly.

Looma Performance (Many-to-One)

Concurrent setting

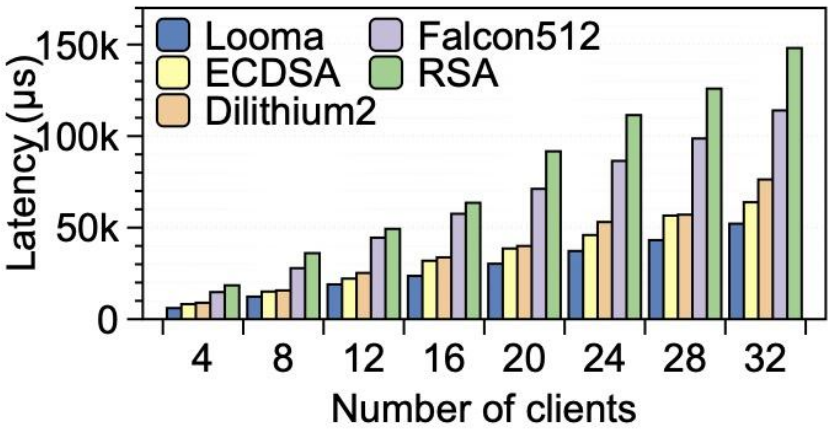


Fig3. mTLS handshake latency p99

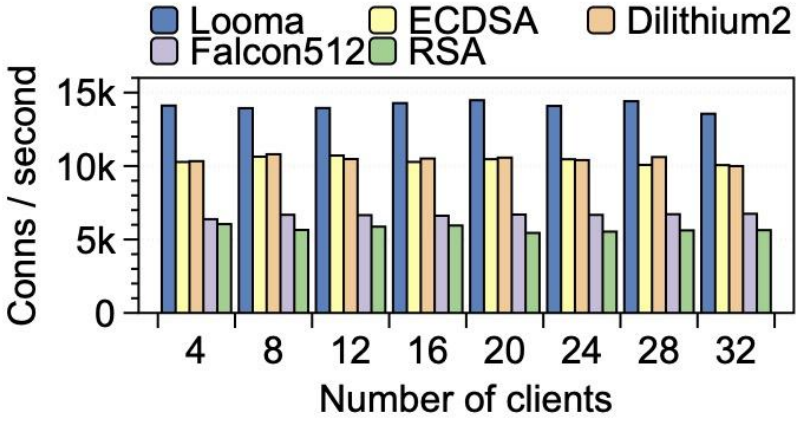


Fig4. mTLS handshake throughput

Takeaways & Questions for CFRG

- Online/offline signatures can reduce PQ authentication latency in TLS-like settings
- Looma is a general approach compatible with multiple PQ signature schemes.

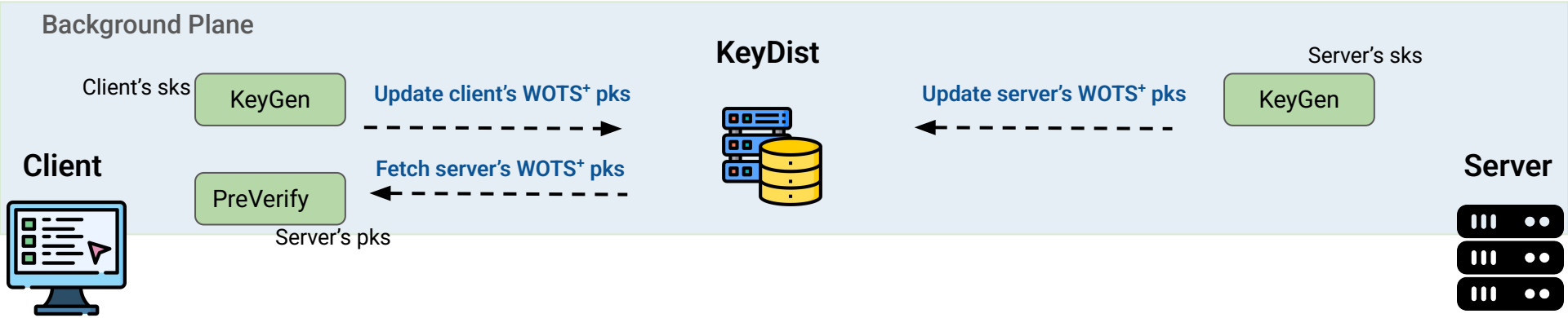
- Our Questions:
  - Are we using the underlying primitives correctly?
    - WOTS^{+}, Merkle trees, Online/offline composition
  - Do the security properties we rely on actually hold?
    - Are there subtle attacks we may have missed?
  - Are there better constructions we should consider?



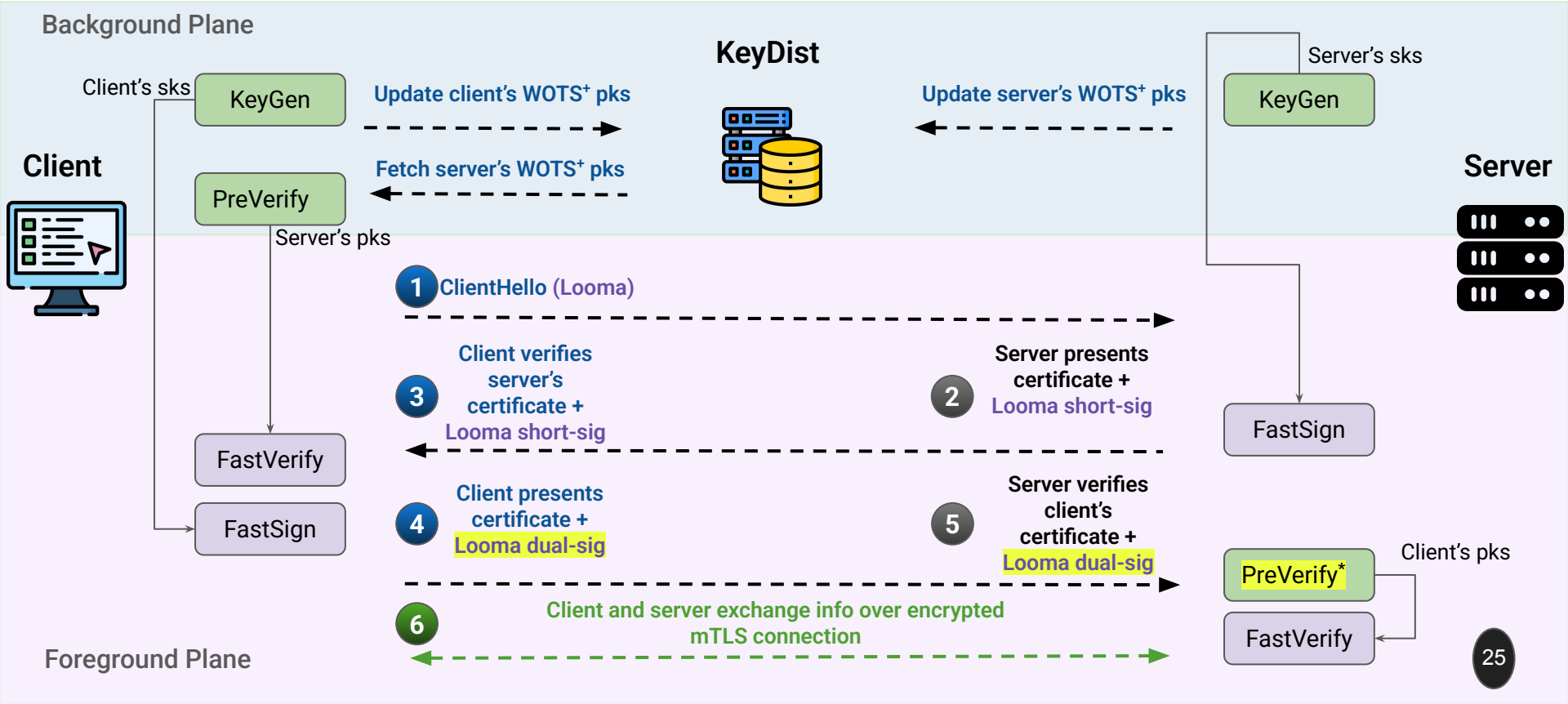
Check out our paper and code!

Backup

What if server has no verification key?

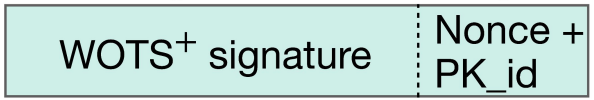


Fallback Strategy 1: Dual-sig mode



Short-sig V.S. Dual-Sig

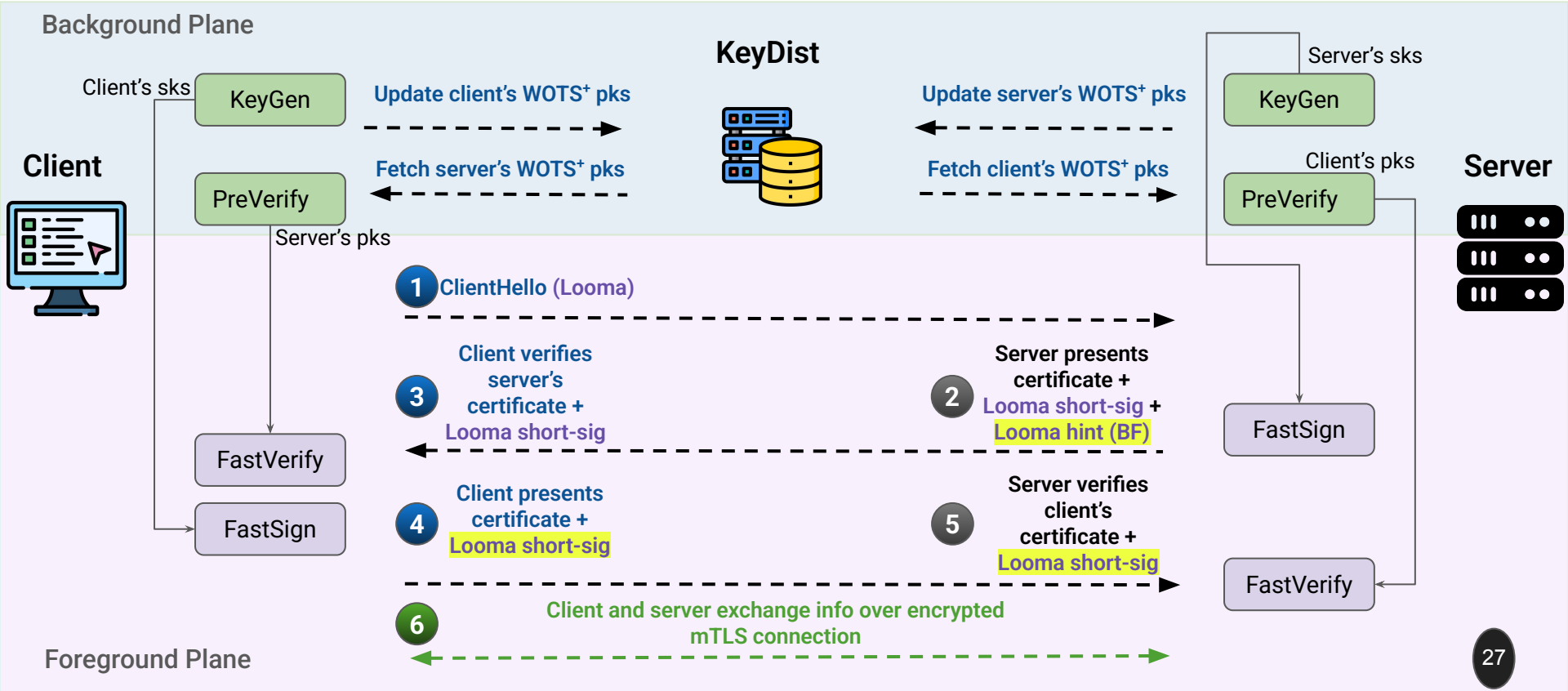
Looma signature — short-sig



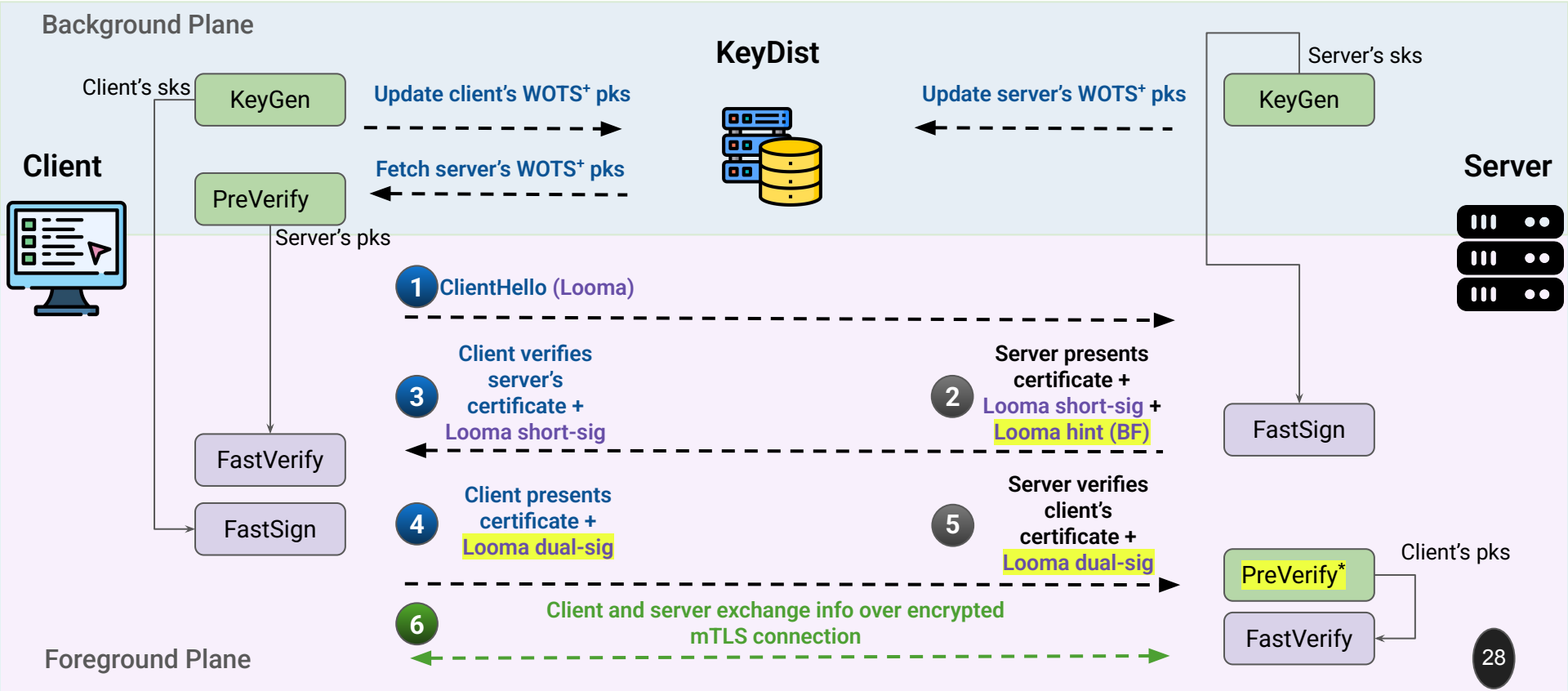
Looma signature — dual-sig



Fallback Strategy 2: Hybrid-Hit mode



Fallback Strategy 2: Hybrid-Miss mode



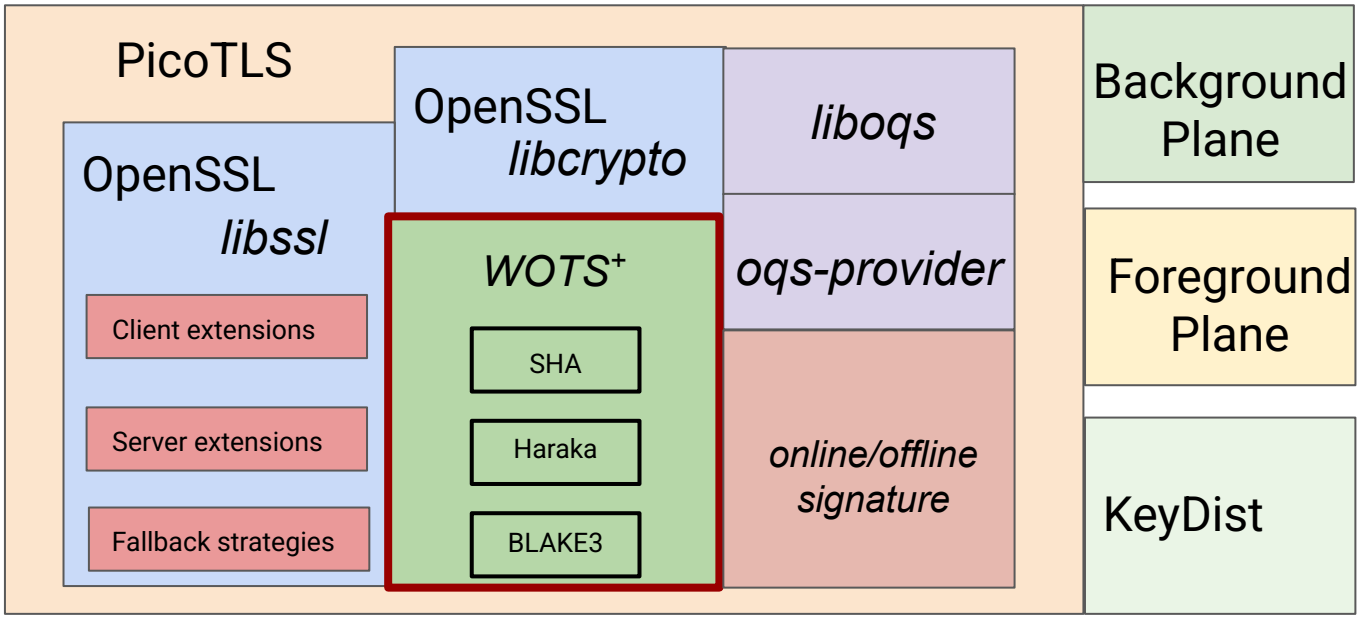
Looma: Implementation Optimizations

TABLE IV: Signature size and per-operation latency of WOTS$^+$ signature across three modern hash families.

w	ℓ	Sig/PK/SK size (B)	Key-gen (µs)				Sign (µs)	Verify (µs)			
			SHA256	BLAKE3	Haraka	Haraka8x		SHA256	BLAKE3	Haraka	Haraka8x
2	265	8,480	35.57	47.23	11.90	7.18	0.62	20.45	23.12	6.43	4.60
4	133	4,256	51.17	68.60	15.26	10.10	0.38	26.82	36.33	8.64	6.43
8	90	2,880	79.42	106.49	22.94	15.42	0.29	72.86	95.33	19.93	13.98
16	67	2,144	128.01	169.33	35.73	24.61	0.25	58.89	83.77	17.16	13.50
32	55	1,760	213.03	290.00	60.45	43.09	0.24	198.58	271.58	52.60	38.63
64	45	1,440	353.73	475.80	99.21	71.07	0.21	308.46	434.51	97.51	70.12

- ➔ WOTS$^+$ sign optimization
  - ◆ Caching intermediate hashes further speeds up FastSign
- ➔ Choose a fast hash
  - ◆ Eight-way SIMD-accelerated Haraka (Haraka8x)
- ➔ Pick a good Winternitz Parameter (w)
  - ◆ w = 4

Looma: Implementation



Instantiation: Dilithium-2 + WOTS$^+$

PQ Authentication Overhead

Example: TLS handshake breakdown

Operation	Algorithm	Latency (µs)	PQ	Library
Keygen+ex	ECDHE[a]	45.20	○	OpenSSL
	KEM[b]	74.70	●	OQS
Verify cert	RSA-2048	17.22	○	OpenSSL
PQ	Dilithium-2	26.27	●	OQS
PQ	Falcon-512	30.7	●	OQS
Sign	ECDSA	14.92	○	OpenSSL
	RSA-2048	200.37	○	OpenSSL
PQ	Dilithium-2	51.74	●	OQS
PQ	Falcon-512	149.2	●	OQS
Verify	ECDSA	40.18	○	OpenSSL
	RSA-2048	15.32	○	OpenSSL
PQ	Dilithium-2	21.27	●	OQS
PQ	Falcon-512	28.3	●	OQS

Mixed Certificate Chains [AsiaCCS'22]

Existing Solutions

Mostly targets key exchange, not authentication.

A. KEMTLS [CCS'20]
   Targets bandwidth reduction as PQ signature is huge

B. OpenSSLNTRU [S&P'22]
   Focus on PQ key exchange acceleration

C. Mixed Certificate Chains [AsiaCCS'22]
   Improves the Certificate verification.

Problem 2: PQTLS Amplifies the Pain

The crypto shift is coming: TLS → Post-Quantum TLS

