PQConnect
Automated Post-Quantum End-to-End Tunnels

Daniel J. Bernstein, Tanja Lange, Jonathan Levin, Bo-Yin Yang

18 November 2025

2024 EU PQC transition roadmap (link)

COMMISSION RECOMMENDATION

of 11.4.2024

on a Coordinated Implementation Roadmap for the transition to Post-Quantum Cryptography

(5) Member States should consider migrating their current digital infrastructures and services for public administrations and other critical infrastructures to Post-Quantum Cryptography as soon as possible, inducing a fundamental shift in cryptographic algorithms, protocols and systems. As highlighted in the Commission's recent White Paper "How to master Europe's digital infrastructure needs", this requires a coordinated effort involving government agencies, standardization bodies, industry stakeholders, researchers and cybersecurity professionals.

(9) Member States and the Union should continue to cooperate actively with their international strategic partners in the development of international standards in Post-Quantum Cryptography with a view to ensuring interoperability of communications going forward.

27 Nov 2024: Joint statement by 17 EU countries (link)

Securing Tomorrow, Today: Transitioning to Post-Quantum Cryptography

> To this end, a Work Stream on PQC, co-chaired by France, Germany and the Netherlands, has been created as part of the NIS Cooperation Group following a recommendation [9] of the European Commission. **We encourage active engagement from all EU member states in this work stream** throughout the process of preparing a roadmap for the transition to Post-Quantum Cryptography to ensure the quantum resilience of the European Union's digital infrastructures.

[..]

27 Nov 2024: Joint statement by 17 EU countries (link)

Securing Tomorrow, Today: Transitioning to Post-Quantum Cryptography

> To this end, a Work Stream on PQC, co-chaired by France, Germany and the Netherlands, has been created as part of the NIS Cooperation Group following a recommendation [9] of the European Commission. **We encourage active engagement from all EU member states in this work stream** throughout the process of preparing a roadmap for the transition to Post-Quantum Cryptography to ensure the quantum resilience of the European Union's digital infrastructures.

[..]

we recommend that these should be protected against 'store now, decrypt later' attacks as soon as possible, latest by the end of 2030. Moreover, we also recommend to develop detailed transition plans for public-key infrastructure systems in the same timeframe.

23 Jun 2025. 1st NIS2 PQC workstream roadmap (link)

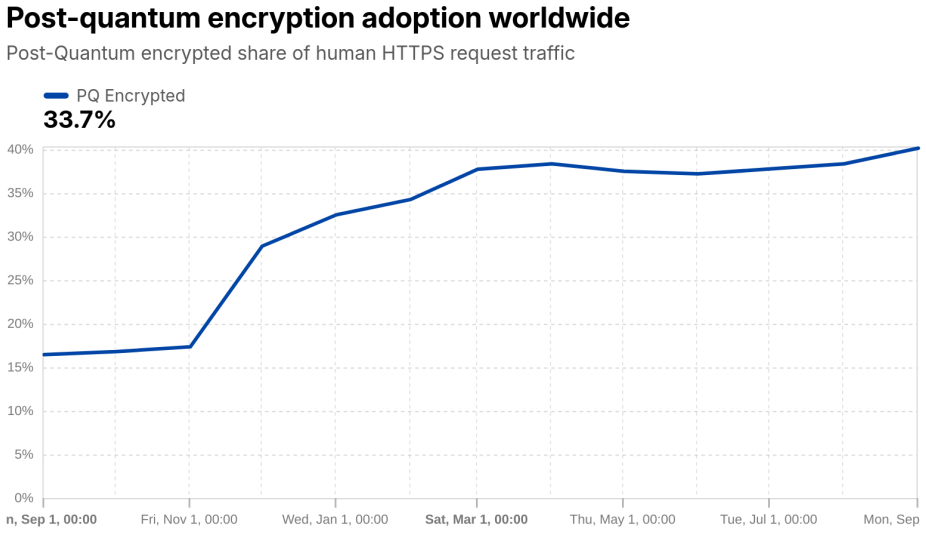
A Coordinated Implementation Roadmap for the Transition to Post-Quantum Cryptography

The EU Member States, supported by the Commission, issued a roadmap and timeline to start using a more complex form of cybersecurity, the so-called post-quantum cryptography (PQC).

Quantum computing has been identified as a threat to many cryptographic algorithms used to protect the confidentiality and authenticity of data. This threat can be countered by a timely, comprehensive and coordinated transition to Post-Quantum Cryptography (PCQ).

AdobeStock © ipopba

Store now, decrypt later



Picture from Cloudflare Radar

But . . . can we do more?

But . . . can we do more?

Without needing to upgrade every application?

PQConnect

- Do not patch PQC onto existing network protocols, but add a new layer with superior security.
- Can be gradually deployed – do your part today!
- Install PQConnect to add support for VPN-like tunnels to your laptop. If you are a system administrator, install it on your servers.
- Every connection will be automatically protected if the laptop and server support PQConnect thanks to automatic peer discovery.
- PQConnect is designed for security. Handshake proven using Tamarin prover (formal verification tool).
- Use Curve25519 (pre-quantum) and Classic McEliece (conservative PQC) for long-term identity keys.
- Use Curve25519 (pre-quantum) and Streamlined NTRU Prime (PQC) for ephemeral keys.

Excursion: VPNs

VPNs

VPNs have a big software-engineering advantage:

VPNs

VPNs have a big software-engineering advantage:

Applications are protected automatically, without modification.

VPNs

VPNs have a big software-engineering advantage:

Applications are protected automatically, without modification.

$\rightsquigarrow$ PQ VPNs can add protection to applications now while waiting for PQ-TLS!

VPNs

Some already use post-quantum cryptography, such as:

- Mullvad,
- Rosenpass, and
- OpenSSH-based VPNs (post-quantum by default since 2022; `sntrup761` supported since 9.0; ML-KEM supported since 10.0).

Typical VPN Usage

Typical VPN Usage

1. VPN client routes traffic through encrypted tunnel to *proxy*.

Typical VPN Usage

1. VPN client routes traffic through encrypted tunnel to *proxy*.

   Problem: Usually not end-to-end

Typical VPN Usage

1. VPN client routes traffic through encrypted tunnel to *proxy*.

   Problem: Usually not end-to-end
2. VPN client talks directly through encrypted tunnel to (multiple) *pre-configured* end-points

Typical VPN Usage

1. VPN client routes traffic through encrypted tunnel to *proxy*.

   Problem: Usually not end-to-end
2. VPN client talks directly through encrypted tunnel to (multiple) *pre-configured* end-points

   Problem: Need to configure endpoints in advance

PQ-VPN

PQ-VPN
&& end-to-end?

PQ-VPN
&& end-to-end?
&& automatic peer discovery?

PQConnect

Like a VPN, PQConnect protects network traffic for all applications.

PQConnect

Like a VPN, PQConnect protects network traffic for all applications.

Unlike a VPN, PQConnect *automatically* discovers peers and creates end-to-end post-quantum tunnels with them.

PQConnect configuration

Client: Install PQConnect.

PQConnect configuration

Client: Install PQConnect.

Server: Install PQConnect, and publish an announcement that the server name supports PQConnect.

PQConnect configuration

Client: Install PQConnect.

Server: Install PQConnect, and publish an announcement that the server name supports PQConnect.

 No peer-specific config needed 

PQConnect Software

1. Provides *application-independent* Post-Quantum network data protection
   - Most applications protected with no extra configuration.
   - Integrates seamlessly with protocols higher in network stack (e.g., TLS)
2. Backward-compatible
   - Automatic peer discovery from advertisements sent to users through DNS.
   - PQConnect computers talk to non-PQConnect computers too.
3. Adds extra cryptographic protections where none currently exist
   - Entire packet encryption, providing header confidentiality
   - Applications without TLS now have encrypted traffic.
4. Client and Server software currently available for GNU/Linux

Connection Overview

1. Client discovers server
2. Client obtains and verifies server's long-term and ephemeral keys
3. Client sends 0RTT handshake message to server
4. Client and Server immediately communicate through resulting tunnel

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

PQConnect view:

- Connect to `kpqc.or.kr`; in DNS lookup:
  `kpqc.or.kr. IN A 210.127.208.136`

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

PQConnect view:

- Connect to `kpqc.or.kr`; in DNS lookup:
  `kpqc.or.kr. IN A 210.127.208.136`

  No PQConnect support detected $\Rightarrow$ **accept**.

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

PQConnect view:

- Connect to `kpqc.or.kr`; in DNS lookup:
  `kpqc.or.kr. IN A 210.127.208.136`

  No PQConnect support detected ⇒ **accept**.

- Connect to `www.pqconnect.net`, in DNS lookup:
  `'www.pqconnect.net. IN CNAME pq1[...].pqconnect.net.'`
  `'pq1[...].pqconnect.net. 60 IN A 131.193.32.108`

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

PQConnect view:

- Connect to `kpqc.or.kr`; in DNS lookup:
  `kpqc.or.kr. IN A 210.127.208.136`

  No PQConnect support detected ⇒ **accept**.

- Connect to `www.pqconnect.net`, in DNS lookup:
  ‘`www.pqconnect.net. IN CNAME pq1[...].pqconnect.net.`’
  ‘`pq1[...].pqconnect.net. 60 IN A 131.193.32.108`

  PQConnect support detected ⇒

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

PQConnect view:

- Connect to `kpqc.or.kr`; in DNS lookup:
  `kpqc.or.kr. IN A 210.127.208.136`

  No PQConnect support detected ⇒ **accept**.

- Connect to `www.pqconnect.net`, in DNS lookup:
  `'www.pqconnect.net. IN CNAME pq1[...].pqconnect.net.'`
  `'pq1[...].pqconnect.net. 60 IN A 131.193.32.108`

  PQConnect support detected ⇒

  1. Rewrite with local address routing to PQConnect (e.g., **10.59.0.2**)

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

PQConnect view:

- Connect to `kpqc.or.kr`; in DNS lookup:
  `kpqc.or.kr. IN A 210.127.208.136`

  No PQConnect support detected ⇒ **accept**.

- Connect to `www.pqconnect.net`, in DNS lookup:
  `'www.pqconnect.net. IN CNAME pq1[...].pqconnect.net.'`
  `'pq1[...].pqconnect.net. 60 IN A 131.193.32.108`

  PQConnect support detected ⇒

  1. Rewrite with local address routing to PQConnect (e.g., **10.59.0.2**)
  2. Initiate session with **131.193.32.108**

Data Flow: Server Identification

PQConnect filters incoming DNS responses.

PQConnect view:

- Connect to `kpqc.or.kr`; in DNS lookup:
  `kpqc.or.kr. IN A 210.127.208.136`

  No PQConnect support detected ⇒ **accept**.

- Connect to `www.pqconnect.net`, in DNS lookup:
  `'www.pqconnect.net. IN CNAME pq1[...].pqconnect.net.'`
  `'pq1[...].pqconnect.net. 60 IN A 131.193.32.108`

  PQConnect support detected ⇒

  1. Rewrite with local address routing to PQConnect (e.g., **10.59.0.2**)
  2. Initiate session with **131.193.32.108**
  3. **Accept**.

Capturing Application Traffic

Application view:

Capturing Application Traffic

Application view:

- `('210.127.208.136', 80) = getaddrinfo('kpqc.or.kr', 80)`

Capturing Application Traffic

Application view:

- `('210.127.208.136', 80) = getaddrinfo('kpqc.or.kr', 80)`
  Great. Send TCP handshake to 210.127.208.136

Capturing Application Traffic

Application view:

- `('210.127.208.136', 80) = getaddrinfo('kpqc.or.kr', 80)`
  Great. Send TCP handshake to 210.127.208.136

- `('10.59.0.2', 80) = getaddrinfo('www.pqconnect.net', 80)`

Capturing Application Traffic

Application view:

- `('210.127.208.136', 80) = getaddrinfo('kpqc.or.kr', 80)`
  Great. Send TCP handshake to 210.127.208.136

- `('10.59.0.2', 80) = getaddrinfo('www.pqconnect.net', 80)`
  Great. Send TCP handshake to **10.59.0.2**

Capturing Application Traffic

Application view:

- `('210.127.208.136', 80) = getaddrinfo('kpqc.or.kr', 80)`
  Great. Send TCP handshake to 210.127.208.136

- `('10.59.0.2', 80) = getaddrinfo('www.pqconnect.net', 80)`
  Great. Send TCP handshake to **10.59.0.2**

Connection now routed through PQConnect

Protocol Overview — Public-Key Cryptography

PQConnect establishes each secure channel using four keys:

	Post-quantum	Pre-quantum
Long-term keys	Classic McEliece	X25519
Ephemeral keys	Streamlined NTRU Prime	X25519

- Classic McEliece is the most conservative PQ cryptosystem. Large public keys but small ciphertexts.
- X25519 is established ECDH using Curve25519.
- Streamlined NTRU Prime is a lattice-based cryptosystem that avoids Kyber's patent issues and some potential security issues. Other deployment: IPSec, SSH.

Comparison to how TLS handles secrecy

Two unfortunate aspects of the TLS design:

- Keys are not erased *inside* a TLS session.
  ⇒ TLS applications are pressured to keep sessions short.
  ⇒ Pressure on session-startup cost.
  ⇒ Incentive to reduce post-quantum security levels.
- TLS long-term keys are signature keys rather than encryption keys.
  ⇒ TLS confidentiality comes only from the keys used for forward secrecy.
  ⇒ Reducing security levels for forward secrecy means reducing security levels for *all* TLS confidentiality.

PQConnect fixes both of these issues:

- Keys are erased in at most 2 minutes inside each session.
- Long-term KEM keys provide authentication *and* encryption.

Protocol Overview — Key Distribution — Long-term Keys

PQConnect establishes a secure channel using:

	Post-quantum	Pre-Quantum
Long-term keys	Classic McEliece	X25519
Ephemeral keys	Streamlined NTRU Prime	X25519

Protocol Overview — Key Distribution — Long-term keys

Classic McEliece (mceliece6960119) pk is 1047319 bytes ($>$ DNS packet).
How can long-term keys be reliably transmitted to the client?

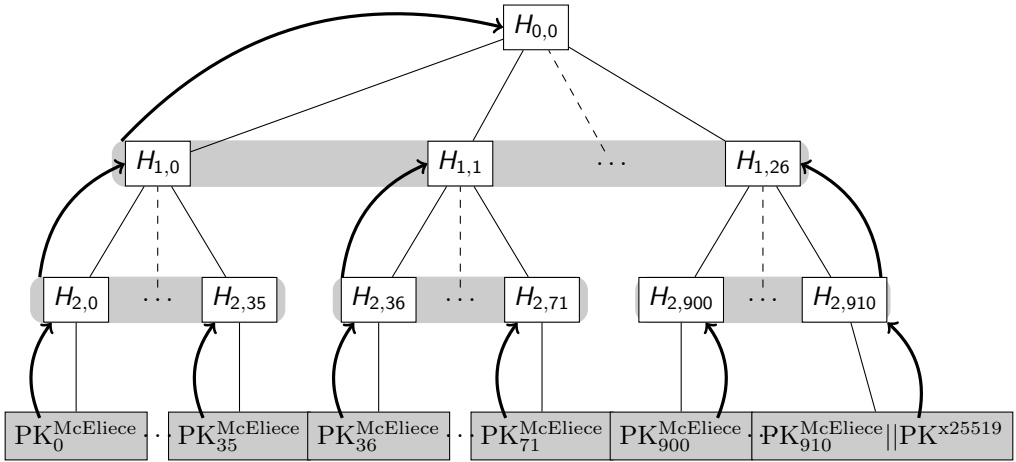
Protocol Overview — Key Distribution — Long-term keys

Classic McEliece (mceliece6960119) pk is 1047319 bytes ($>$ DNS packet).
How can long-term keys be reliably transmitted to the client?

$\Rightarrow$ Distribute long-term keys as a Merkle tree

1. Packet-sized chunks of key material form the leaves
2. Publish the hash at the root of the tree in DNS
3. Clients request the tree level by level, immediately verifying each packet.

Protocol Overview — Key Distribution — Long-term keys



Protocol Overview — Key Distribution — Ephemeral Keys

PQConnect establishes a secure channel using:

	Post-quantum	Pre-Quantum
Long-term keys	Classic McEliece	X25519
Ephemeral keys	Streamlined NTRU Prime	X25519

Protocol Overview — Key Distribution — Ephemeral Keys

Scheme name	PK size
Streamlined NTRU Prime	1158 B
X25519	32 B

Both public keys fit into a single UDP packet (< 1280 B).
Client requests this packet from the server before sending a handshake message.
Keys rotate every 30 seconds (multiple use, short lifespan)
Server will accept ciphertexts for expired keys for another 90s (120s total)

Protocol Overview — Key Exchange

PQConnect establishes a secure channel using:

	Post-quantum	Pre-Quantum
Long-term keys	Classic McEliece	X25519
Ephemeral keys	Streamlined NTRU Prime	X25519

Protocol Overview — Key Exchange

PQConnect establishes a secure channel using:

	Post-quantum	Pre-Quantum
Long-term keys	Classic McEliece	X25519
Ephemeral keys	Streamlined NTRU Prime	X25519

Client computes secret against all four public keys.
Sends handshake M to server.

Server receives M, decapsulates, etc., and computes same shared secret.

Protocol Overview — Key Exchange — Handshake security

Hybrid approach

4 key agreements take place:

- 2 long-term (post- and pre-quantum)
- 2 ephemeral (post- and pre-quantum)

Each PKC scheme layered “inside of” the next.
Forces sequential attack to obtain innermost keys/ciphertexts.

PQ cryptography protects against quantum attackers.
X25519 assures that security is no worse than current state-of-the-art

Protocol Overview — Key Exchange — Handshake security

Hybrid approach

4 key agreements take place:

- 2 long-term (post- and pre-quantum)
- 2 ephemeral (post- and pre-quantum)

Each PKC scheme layered "inside of" the next.
Forces sequential attack to obtain innermost keys/ciphertexts.

PQ cryptography protects against quantum attackers.
X25519 assures that security is no worse than current state-of-the-art

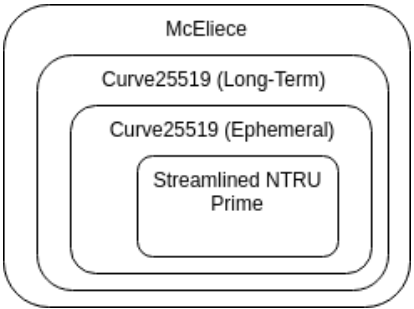
Formal security proof

Security properties of the handshake were formally proven using Tamarin Prover[1].

[1] https://tamarin-prover.github.io/

Protocol Overview — Key Exchange — "Nesting" schemes

Most conservative system on the outside.



Attacker can see and break server's Curve25519 keys with a QC,
but cannot obtain DH secret, as client's share encrypted under Classic McEliece.

Protocol Overview — Symmetric Crypto

The longer a key is in use, the more data can be recovered if it is leaked.

Protocol Overview — Symmetric Crypto

The longer a key is in use, the more data can be recovered if it is leaked.
Each packet sent between peers encrypted with unique key.

Protocol Overview — Symmetric Crypto

The longer a key is in use, the more data can be recovered if it is leaked.
Each packet sent between peers encrypted with unique key.
Packets can be delayed, dropped, reordered.

Protocol Overview — Symmetric Crypto

The longer a key is in use, the more data can be recovered if it is leaked.
Each packet sent between peers encrypted with unique key.
Packets can be delayed, dropped, reordered.
How long to keep keys in memory?

Protocol Overview — Symmetric Crypto

The longer a key is in use, the more data can be recovered if it is leaked.
Each packet sent between peers encrypted with unique key.
Packets can be delayed, dropped, reordered.
How long to keep keys in memory?

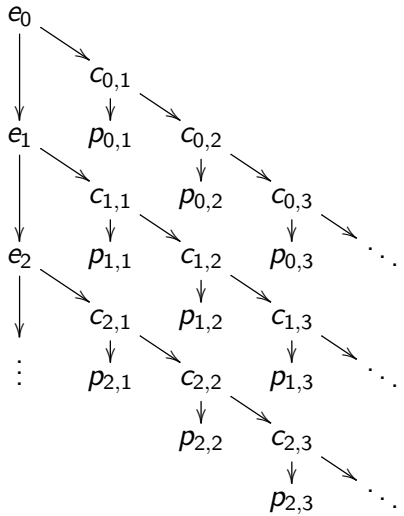
- When sending: erase key immediately after use, ratchet forward using KDF.
- When receiving: erase key immediately after use, and at the latest after 2 min.

Key ratchet advances by message and by time

e_0 is the initial epoch key.
Immediately advance ratchet in 3 ways:

- New epoch key: e_1.
- New chain keys: $c_{0,1}, c_{0,2}$.
- New packet key: $p_{0,1}$.

New epoch every 30 seconds.
Keys erased upon use, or at latest after 2 minutes.
Packet keys allow for delays (see next slide).

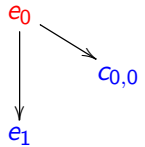


Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 0s$ e_0
Received packets:

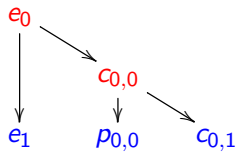
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 0s$
Received packets:



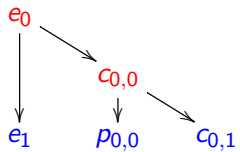
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 0s$
Received packets:



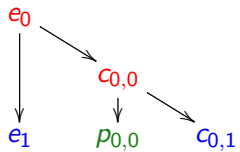
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 1s$
Received packets: $P_{0,0}$



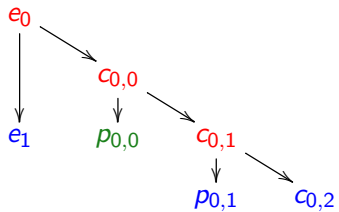
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 1s$
Received packets: $P_{0,0}$



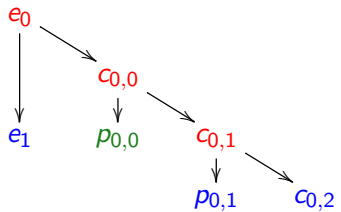
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 1s$
Received packets: $P_{0,0}$



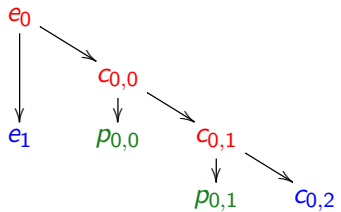
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 2s$
Received packets: $P_{0,0}$ $P_{0,1}$



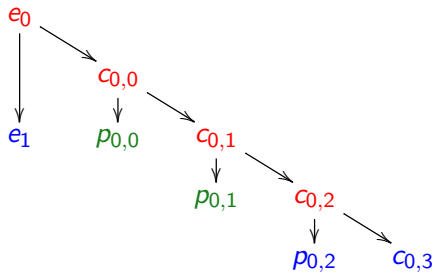
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 2s$
Received packets: $P_{0,0}$ $P_{0,1}$



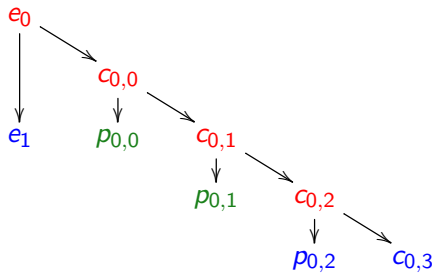
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 2s$
Received packets: $P_{0,0}$ $P_{0,1}$



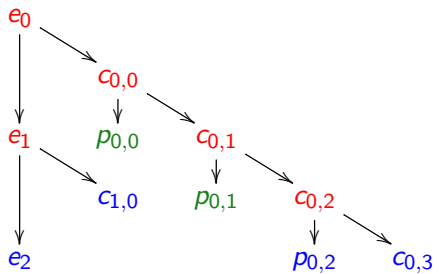
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 115s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$



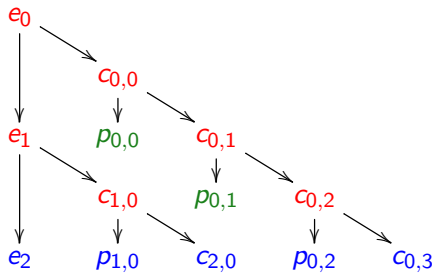
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 115s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$



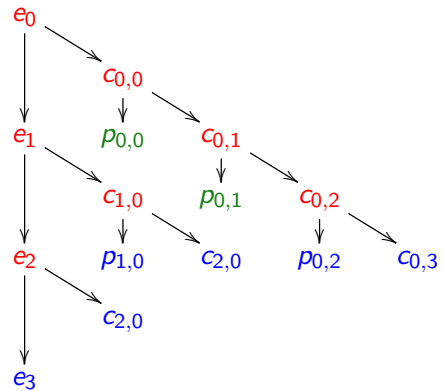
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 115s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$



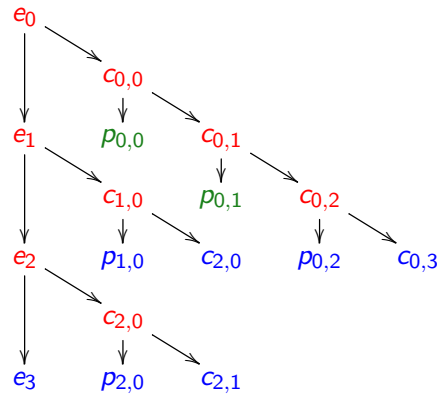
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 115s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$



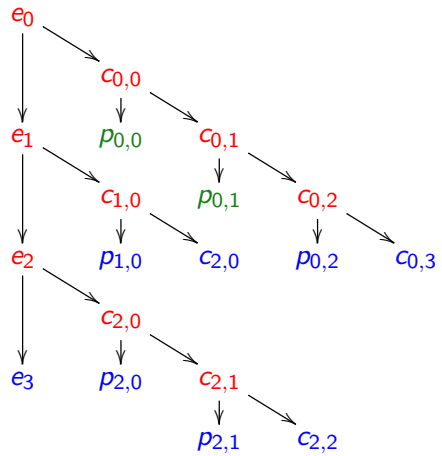
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 115s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$



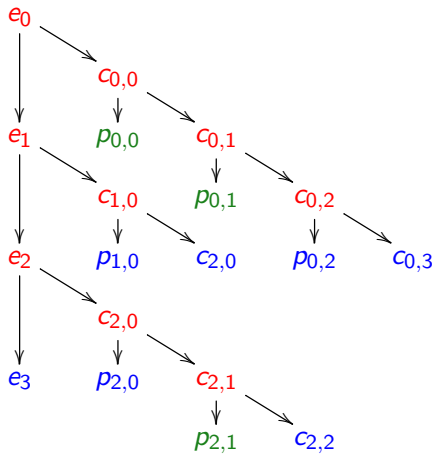
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 115s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$



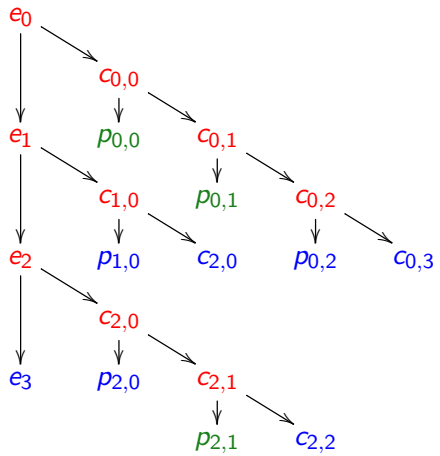
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 115s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$



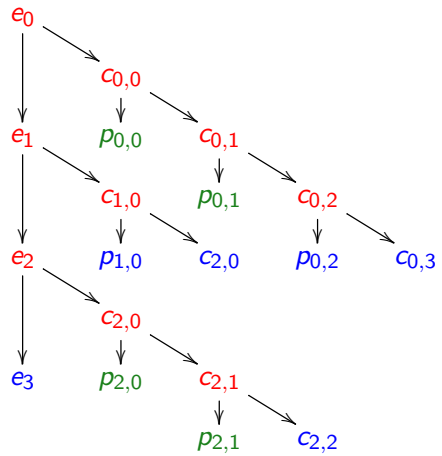
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 116s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$



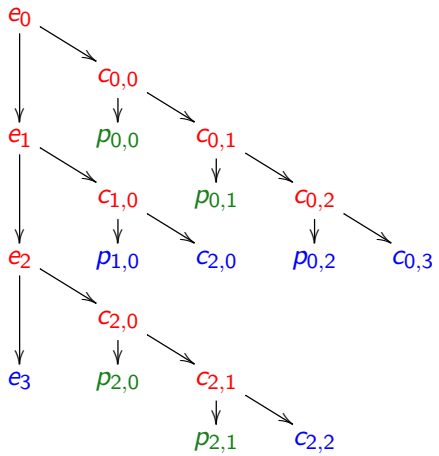
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 116s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$



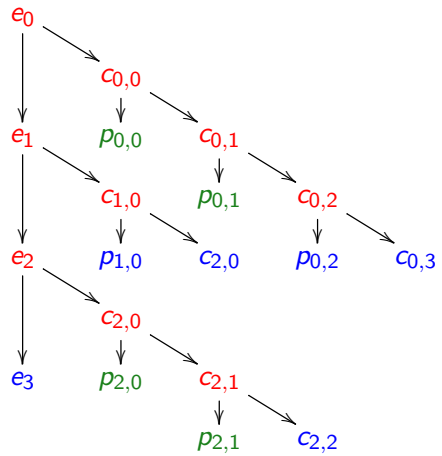
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 117s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$



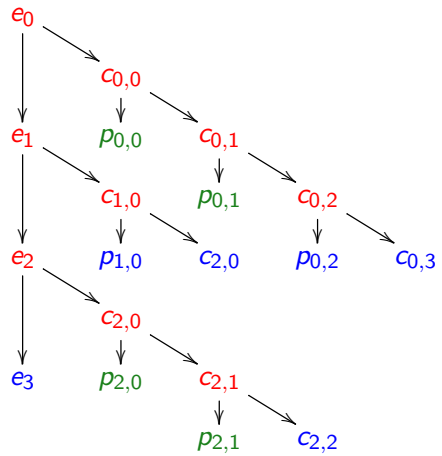
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 118s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$



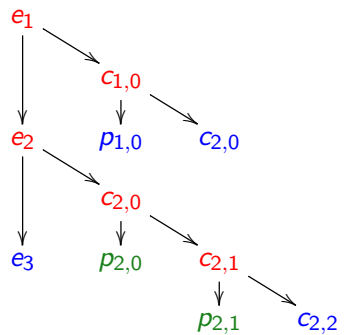
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 119s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$



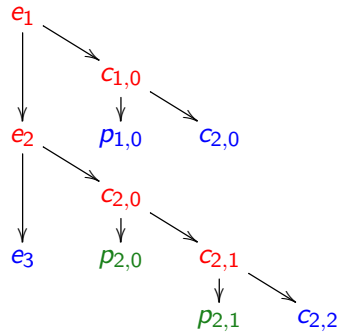
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 120s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$



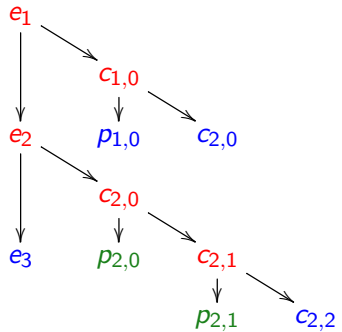
Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 121s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$ $P_{0,2}$



Receiver's Key Ratchet

e_i: Epoch Key; $c_{i,j}$: Chain Key; $p_{i,j}$: Packet Key;
In memory; Erased; Used and Erased
$t = 121s$
Received packets: $P_{0,0}$ $P_{0,1}$ $P_{2,1}$ $P_{2,0}$ $P_{0,2}$



The underlying crypto microlibraries

These are separate software projects (also packaged in Debian)
taking responsibility for particular cryptosystems:

- `https://lib.mceliece.org` for Classic McEliece
- `https://lib25519.cr.yp.to` for Curve25519
- `https://libntruprime.cr.yp.to` for Streamlined NTRU Prime

Written in C and assembly; easy to call from other languages.
Extensive tests, optimizations, protection against timing attacks.

Installing PQConnect

Debian 13:

- **Client:**
  `$ sudo apt install pqconnect`
- **Server:**
  `$ sudo apt install pqconnect-server`

Other OSes:

- **Client:**
  See “Quick start” lines in https://www.pqconnect.net/user.html
- **Server:**
  See “Quick start” lines in https://www.pqconnect.net/sysadmin.html

Modular software ⇒ flexible installation

For a computer that splits services across virtual machines (VMs), can create a VM running PQConnect to protect incoming connections to services on the existing VMs. ("Server-in-a-bottle mode.")

This is how PQConnect service works today for `bench.cr.yp.to`, `csidh.isogeny.org`, `gcd.cr.yp.to`, `lib.mceliece.org`, etc.

VM's sysadmin can run PQConnect for that VM. Example: `www.jlev.in`.

Another example: Can put a router in front of an existing computer, run PQConnect on the router to protect connections to (or from) the computer.

No changes to the application software or application configuration.

More information

- Read the paper, published at NDSS 2025, at https://www.pqconnect.net/pqconnect-20241206.pdf.
- Visit https://www.pqconnect.net/ for more information and software download.
- Join https://zulip.pqconnect.net/ to discuss the project and get updates.