Secure implementations of post-quantum schemes

Gustavo Banegas

Qualcomm

June 3, 2024

Summer School on real-world crypto and privacy

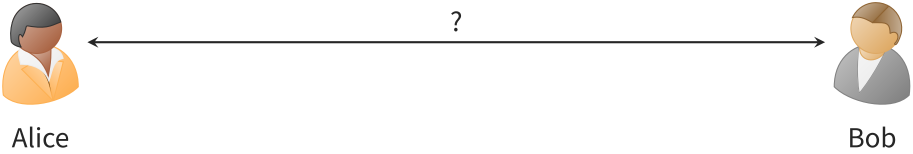
"Food gives you energy"

Me after I eat:

TABLE OF CONTENTS

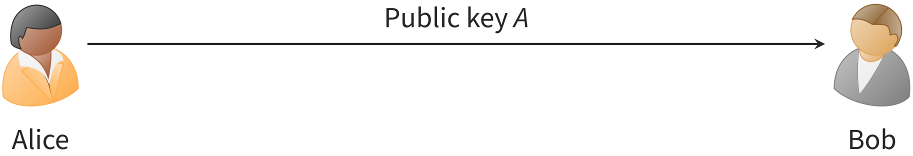
TABLE OF CONTENTS

ESTABLISHING A SECURE CONNECTION

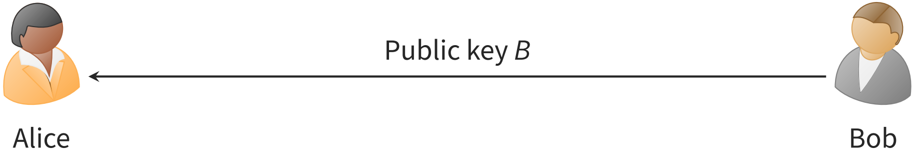


How can we exchange data securely?

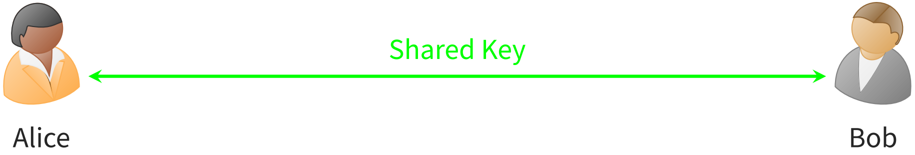
ESTABLISHING A SECURE CONNECTION



ESTABLISHING A SECURE CONNECTION

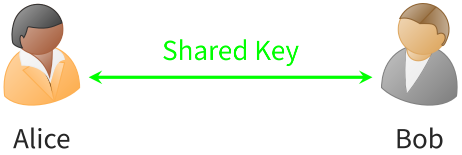


ESTABLISHING A SECURE CONNECTION



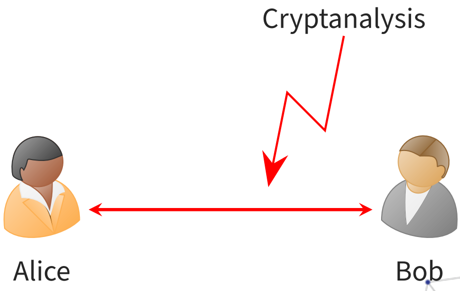
- Key exchange is a fundamental step for establishing a secure connection.
- Cryptographic key agreement schemes can be built from other methods:
  - Diffie-Hellman;
  - RSA;
  - New post-quantum algorithms.

THREATS TO CRYPTOGRAPHY



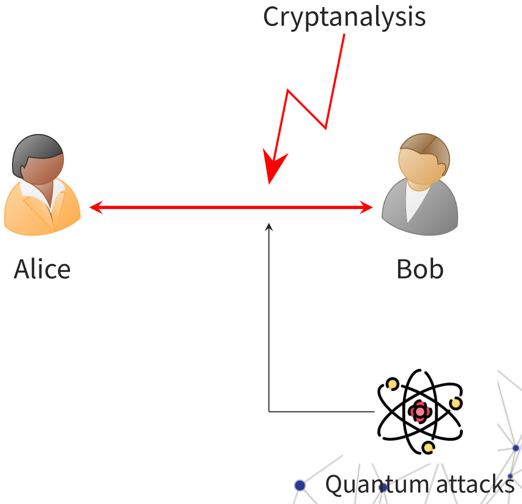
THREATS TO CRYPTOGRAPHY

- **Cryptanalysis**:
  - Exploit mathematical and algorithmic weaknesses.
-
-



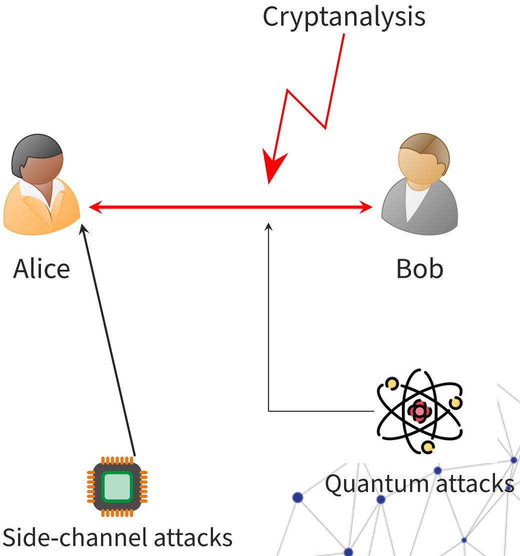
THREATS TO CRYPTOGRAPHY

- **Cryptanalysis**:
  - Exploit mathematical and algorithmic weaknesses.
- **Quantum Attacks**:
  - Exploit vulnerabilities to quantum computation.



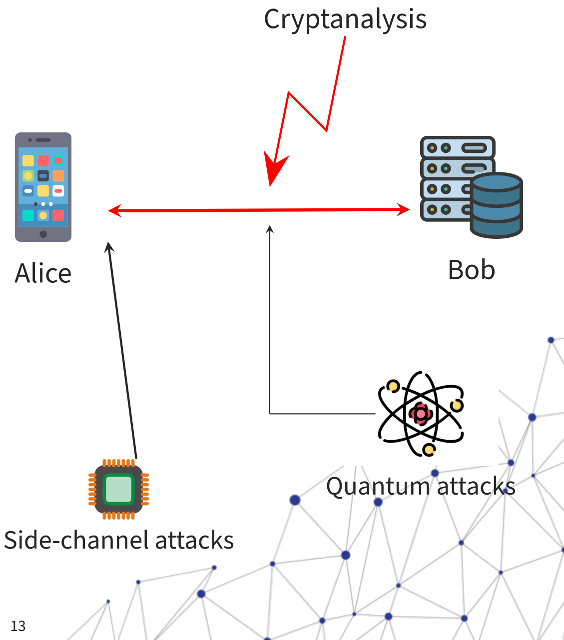
THREATS TO CRYPTOGRAPHY

- **Cryptanalysis**:
  - Exploit mathematical and algorithmic weaknesses.
- **Quantum Attacks**:
  - Exploit vulnerabilities to quantum computation.
- **Side-Channel Attacks**:
  - Exploit runtime information:
    - + timing (local and remote);
    - + power consumption;
    - + electromagnetic radiation.



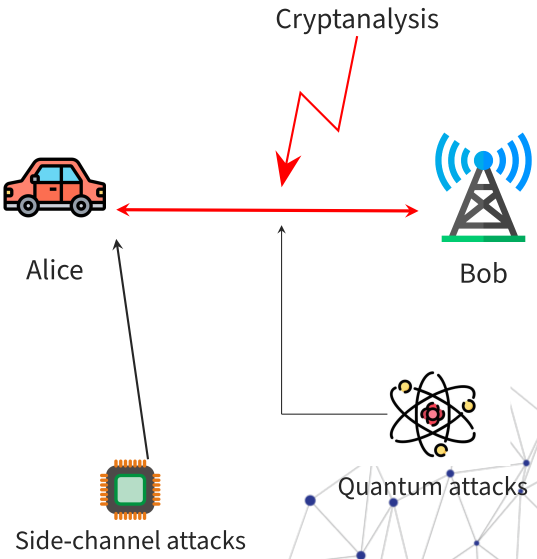
THREATS TO CRYPTOGRAPHY

- **Cryptanalysis**:
  - Exploit mathematical and algorithmic weaknesses.
- **Quantum Attacks**:
  - Exploit vulnerabilities to quantum computation.
- **Side-Channel Attacks**:
  - Exploit runtime information:
    - + timing (local and remote);
    - + power consumption;
    - + electromagnetic radiation.



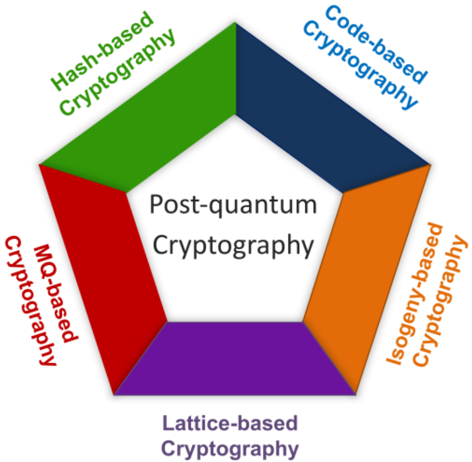
THREATS TO CRYPTOGRAPHY

- **Cryptanalysis**:
  - Exploit mathematical and algorithmic weaknesses.
- **Quantum Attacks**:
  - Exploit vulnerabilities to quantum computation.
- **Side-Channel Attacks**:
  - Exploit runtime information:
    + timing (local and remote);
    + power consumption;
    + electromagnetic radiation.



POST-QUANTUM CRYPTOGRAPHY

Post-quantum cryptography overview:



POST-QUANTUM CRYPTOGRAPHY AND NIST CALL

- 2017: NIST launched call to post-quantum standards to replace RSA/ECC;
- 2022: NIST selects 1 key exchange mechanism and 3 signatures;
    - Kyber as key exchange;
    - Dilithium, Sphincs+, and Falcon as digital signatures;
    - Further analysis: HQC, Bike and McEliece;
- 2023: NIST opened new call for signatures.

WHAT IS A SECURE IMPLEMENTATION?

It does not leak:

- Time information;
- Power consumption;
- or any secret data.

HOW DO WE ACHIEVE A SECURE IMPLEMENTATION?

- Constant-time implementation:

HOW DO WE ACHIEVE A SECURE IMPLEMENTATION?

- Constant-time implementation:
  - ***Constant-time property does not mean that time is deterministic***;
  - *It is constant-time if the algorithm time provides no information about the input.*

HOW DO WE ACHIEVE A SECURE IMPLEMENTATION?

- Constant-time implementation:
  - ***Constant-time property does not mean that time is deterministic***;
  - *It is constant-time if the algorithm time provides no information about the input.*
- Masking:
  - *Combine random values (masks) with the input*;

HOW DO WE ACHIEVE A SECURE IMPLEMENTATION?

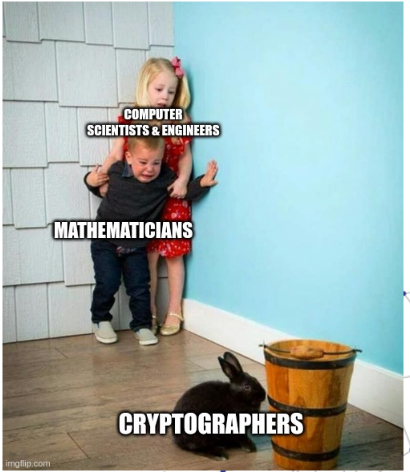
- Constant-time implementation:
  - ***Constant-time property does not mean that time is deterministic***;
  - *It is constant-time if the algorithm time provides no information about the input.*
- Masking:
  - *Combine random values (masks) with the input*;
- others more specific:
  - Blinding;
  - Shuffling;
  - Random order execution, and etc.

SELECT YOUR THREAT MODEL

- Who are you against?
-
-

SELECT YOUR THREAT MODEL

- Who are you against?
-
-



SELECT YOUR THREAT MODEL

- Who are you against?
- Where the code will be use?
-

SELECT YOUR THREAT MODEL

- Who are you against?
- Where the code will be use?
-

SELECT YOUR THREAT MODEL

- Who are you against?
- Where the code will be use?
-

SELECT YOUR THREAT MODEL

- Who are you against?
- Where the code will be use?
- What is your application?

WHAT IS A SYSTEM-ON-CHIP?

It is easier to explain what a System-On-Chip (SoC) contains:

- central processing unit (CPU);

WHAT IS A SYSTEM-ON-CHIP?

It is easier to explain what a System-On-Chip (SoC) contains:

- central processing unit (CPU);
- memory interfaces;
-
-
-
-

WHAT IS A SYSTEM-ON-CHIP?

It is easier to explain what a System-On-Chip (SoC) contains:

- central processing unit (CPU);
- memory interfaces;
- input/output devices and interfaces;
-
-
-

WHAT IS A SYSTEM-ON-CHIP?

It is easier to explain what a System-On-Chip (SoC) contains:

- central processing unit (CPU);
- memory interfaces;
- input/output devices and interfaces;
- secondary storage interfaces;
-
-

WHAT IS A SYSTEM-ON-CHIP?

It is easier to explain what a System-On-Chip (SoC) contains:

- central processing unit (CPU);
- memory interfaces;
- input/output devices and interfaces;
- secondary storage interfaces;
- graphics processing unit (GPU);

WHAT IS A SYSTEM-ON-CHIP?

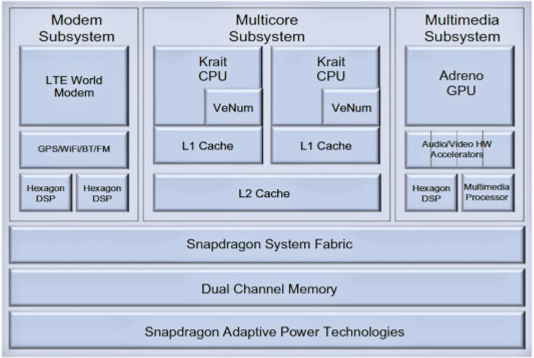
It is easier to explain what a System-On-Chip (SoC) contains:

- central processing unit (CPU);
- memory interfaces;
- input/output devices and interfaces;
- secondary storage interfaces;
- graphics processing unit (GPU);
- Other components.

WHAT IS A SYSTEM-ON-CHIP?

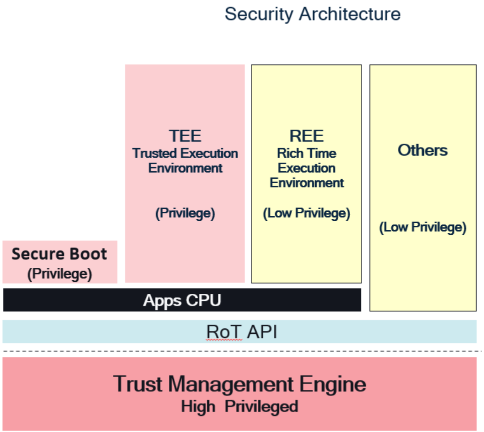
It is easier to explain what a System-On-Chip (SoC) contains:

- central processing unit (CPU);
- memory interfaces;
- input/output devices and interfaces;
- secondary storage interfaces;
- graphics processing unit (GPU);
- Other components.



HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

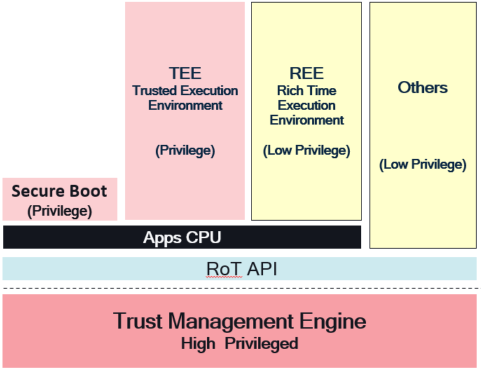
The anatomy of a System on Chip



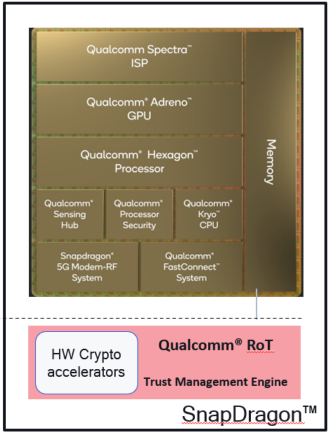
HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

Security Architecture



SoC Architecture



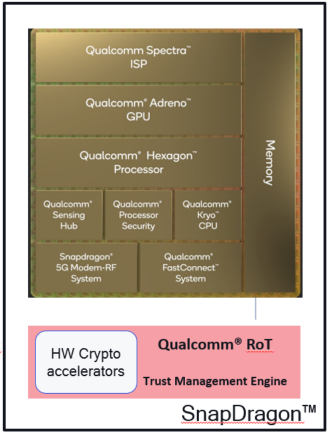
SoC main High Compute CPU hosting HLOS (Android, Windows..) is called

Apps CPU

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto

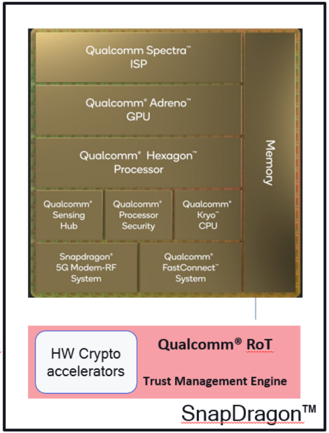


Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based				
HW Crypto Accelerators /Co-processor				

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip



RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto

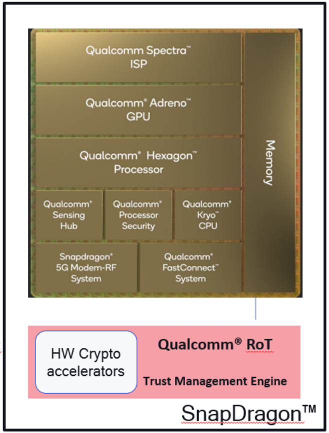
Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes			
HW Crypto Accelerators /Co-processor				

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto



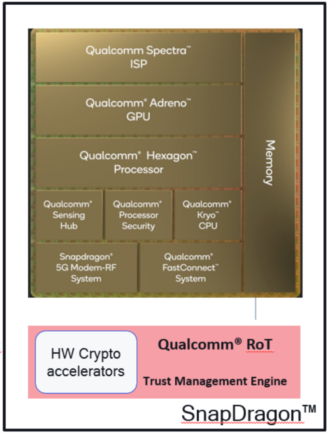
Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes			
HW Crypto Accelerators /Co-processor	Implementation Dependant			

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto



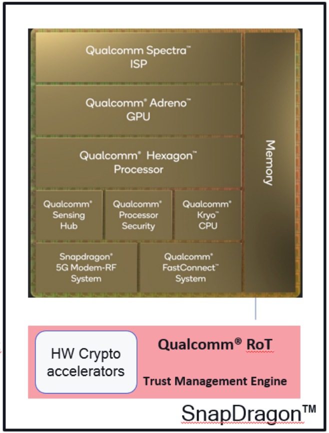
Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes	No		
HW Crypto Accelerators /Co-processor	Implementation Dependant			

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto



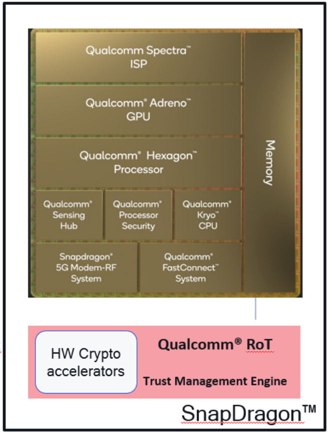
Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes	No		
HW Crypto Accelerators /Co-processor	Implementation Dependant	Yes		

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto



Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes	No	No: Key exposure	
HW Crypto Accelerators /Co-processor	Implementation Dependant	Yes		

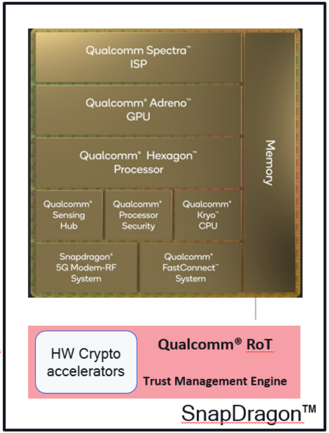
Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions

- Lifecycle Mgt
- Key Mgt
- Crypto



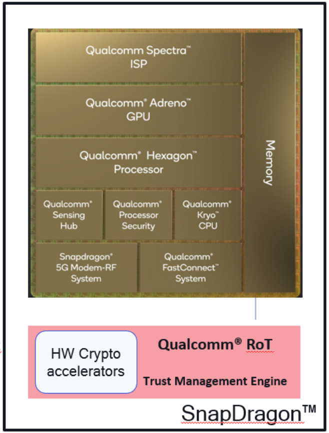
Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes	No	No: Key exposure	
HW Crypto Accelerators /Co-processor	Implementation Dependant	Yes	Enabled	

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto



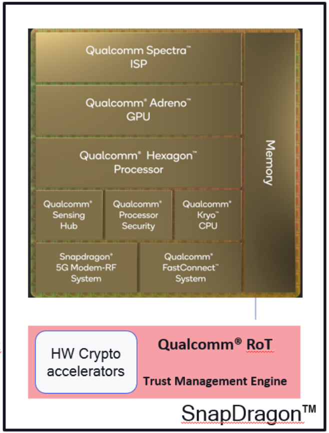
Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes	No	No: Key exposure	Yes
HW Crypto Accelerators /Co-processor	Implementation Dependant	Yes	Enabled	

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

HOW IS A SOC AND CRYPTOGRAPHY IN PRACTICE?

The anatomy of a System on Chip

RoT functions
- Lifecycle Mgt
- Key Mgt
- Crypto



Criteria / Crypto Processing	Performance / Cost	CPU Offload/Power	Security and Integrity	Agility
App CPU based	Yes	No	No: Key exposure	Yes
HW Crypto Accelerators /Co-processor	Implementation Dependant	Yes	Enabled	Limited

Security Mandate dedicated HW Crypto Accelerator / coprocessor approach

TABLE OF CONTENTS

VALGRIND

If you code in C, Valgrind is your best friend.

- Dynamic analysis tools;
- Memory leak;
- Profiler tool;

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;

- No memory access based on secret-dependent values;

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;
- No secret-dependent values given to some variable time functions;
-
-

- No memory access based on secret-dependent values;
-
-
-

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;
- No secret-dependent values given to some variable time functions;

- No memory access based on secret-dependent values;
- No secret-dependent values given to some variable time functions;

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;
- No secret-dependent values given to some variable time functions;
- Avoid data-dependent instruction timing;
-

- No memory access based on secret-dependent values;
- No secret-dependent values given to some variable time functions;
-
-

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;
- No secret-dependent values given to some variable time functions;
- Avoid data-dependent instruction timing;
-

- No memory access based on secret-dependent values;
- No secret-dependent values given to some variable time functions;
- Use fixed-time lookup tables;
-

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;
- No secret-dependent values given to some variable time functions;
- Avoid data-dependent instruction timing;
- Use constant-time arithmetic operations;

- No memory access based on secret-dependent values;
- No secret-dependent values given to some variable time functions;
- Use fixed-time lookup tables;
-

VALGRIND AND CONSTANT-TIME

- No branching on secret-dependent values;
- No secret-dependent values given to some variable time functions;
- Avoid data-dependent instruction timing;
- Use constant-time arithmetic operations;

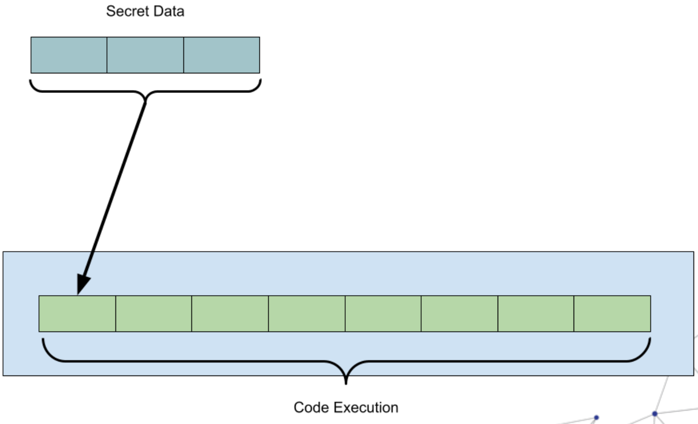
- No memory access based on secret-dependent values;
- No secret-dependent values given to some variable time functions;
- Use fixed-time lookup tables;
- Ensure no compiler optimizations introduce timing variability.

VALGRIND CONSTANT TIME VERIFICATION

- We “poison” the secret data, that is, we put an undefined value;
- *valgrind* will check if the undefined data corrupts branches or indices.

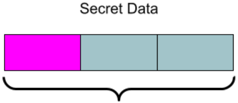
HOW TO USE *valgrind* TO CHECK SENSITIVE DATA

Correct flow without “poisoning”:



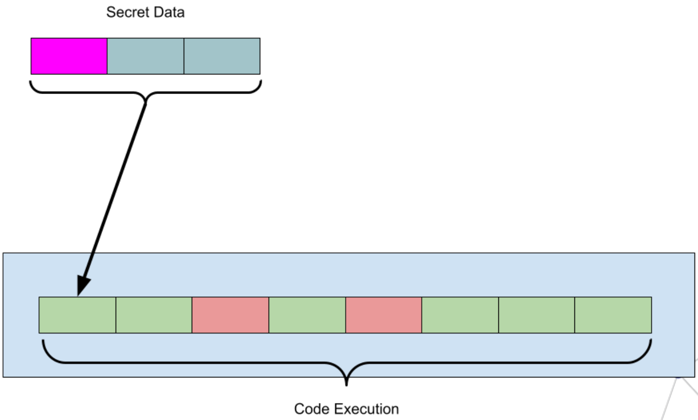
HOW TO USE *valgrind* TO CHECK SENSITIVE DATA

We poison the secret data with "undefined" value



HOW TO USE *valgrind* TO CHECK SENSITIVE DATA

We check where "undefined" value impacts in the code execution



we used *valgrind* to check if poisoning the secret data generates leaks of sensitive information

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

If you want to use it: `https://github.com/gbanegas/class_ct`

complete version in: `https://neuromancer.sk/article/29`

```c
#include <memcheck.h>
/*
   Use this function to mark any memory
       regions containing secret data.
 */
#define poison(addr, len)
    VALGRIND_MAKE_MEM_UNDEFINED(addr,
    len)
```

```c
int modulus = 65535;
int base = 123;
int exponent = 981357566;
// mark the exponent as secret
poison(&exponent, sizeof(int));
int res = modular_pow(base, exponent,
     modulus);
```

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

If you want to use it: `https://github.com/gbanegas/class_ct`

complete version in: `https://neuromancer.sk/article/29`

```c
#include <memcheck.h>
/*
   Use this function to mark any memory
        regions containing secret data.
 */
#define poison(addr, len)
    VALGRIND_MAKE_MEM_UNDEFINED(addr,
    len)
```

```c
int modulus = 65535;
int base = 123;
int exponent = 981357566;
// mark the exponent as secret
poison(&exponent, sizeof(int));
int res = modular_pow(base, exponent,
     modulus);
```

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

If you want to use it: `https://github.com/gbanegas/class_ct`

complete version in: `https://neuromancer.sk/article/29`

```c
#include <memcheck.h>
/*
   Use this function to mark any memory
        regions containing secret data.
 */
#define poison(addr, len)
    VALGRIND_MAKE_MEM_UNDEFINED(addr,
    len)
```

```c
int modulus = 65535;
int base = 123;
int exponent = 981357566;
// mark the exponent as secret
poison(&exponent, sizeof(int));
int res = modular_pow(base, exponent,
     modulus);
```

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

```
==9133== Conditional jump or move depends on uninitialised value(s)
==9133==    at 0x4004F8: modular_pow (example.c:10)
==9133==    by 0x4005DF: main (example.c:28)
==9133==  Uninitialised value was created by a client request
==9133==    at 0x4005C6: main (example.c:25)
==9133==
```

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

```
int modular_pow(int base, int exponent, int modulus) {
  if(modulus == 1) {
    return 0;
  }

  int result = 1;
  base = base % modulus;
  while(exponent > 0) {
    if (exponent % 2 == 1) {
      result = (result * base) % modulus;
    }
    exponent = exponent >> 1;
    base = (base * base) % modulus;
  }
  return result;
}
```

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

```c
int modular_pow(int base, int exponent, int modulus) {
  if(modulus == 1) {
    return 0;
  }

  int result = 1;
  base = base % modulus;
  while(exponent > 0) {
    if (exponent % 2 == 1) {
      result = (result * base) % modulus;
    }
    exponent = exponent >> 1;
    base = (base * base) % modulus;
  }
  return result;
}
```

```
==9133== Conditional jump or move depends on uninitialised value(s)
==9133==    at 0x4004F8: modular_pow (example.c:10)
==9133==    by 0x4005DF: main (example.c:28)
==9133==  Uninitialised value was created by a client request
==9133==    at 0x4005C6: main (example.c:25)
==9133==
```

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

```
int modular_pow(int base, int exponent, int modulus) {
  if(modulus == 1) {
    return 0;
  }

  int result = 1;
  base = base % modulus;
  while(exponent > 0) {
    if (exponent % 2 == 1) {
      result = (result * base) % modulus;
    }
    exponent = exponent >> 1;
    base = (base * base) % modulus;
  }
  return result;
}
```

```
==9133== Conditional jump or move depends on uninitialised value(s)
==9133==    at 0x4004F8: modular_pow (example.c:10)
==9133==    by 0x4005DF: main (example.c:28)
==9133==  Uninitialised value was created by a client request
==9133==    at 0x4005C6: main (example.c:25)
==9133==
```

HOW TO USE VALGRIND FOR CHECKING CONSTANT-TIME?

```c
int modular_pow(int base, int exponent, int modulus) {
  if(modulus == 1) {
    return 0;
  }

  int result = 1;
  base = base % modulus;
  while(exponent > 0) {
    if (exponent % 2 == 1) {
      result = (result * base) % modulus;
    }
    exponent = exponent >> 1;
    base = (base * base) % modulus;
  }
  return result;
}
```

```
==9133== Conditional jump or move depends on uninitialised value(s)
==9133==    at 0x4004F8: modular_pow (example.c:10)
==9133==    by 0x4005DF: main (example.c:28)
==9133==  Uninitialised value was created by a client request
==9133==    at 0x4005C6: main (example.c:25)
==9133==
```

TABLE OF CONTENTS

ELLIPTIC CURVES AND ISOGENIES

- An elliptic curve E over a finite field $\mathbb{F}_p$ is given by the equation:

$$y^2 = x^3 + ax + b$$

- An isogeny $\phi : E \to E'$ is a non-constant rational map that preserves the group structure.
- For supersingular elliptic curves, the endomorphism ring is isomorphic to a maximal order in a quaternion algebra.

CSIDH

CSIDH is a post-quantum isogeny-based non-interactive key exchange protocol.

It uses a group action on a certain set of elliptic curves.

CSIDH

CSIDH is a post-quantum isogeny-based non-interactive key exchange protocol.

It uses a group action on a certain set of elliptic curves.

- Secret keys sampled from some keyspace $\mathrm{sk} \in \mathcal{K}$ give group elements,
- Public keys are elliptic curves obtained by evaluating the group action $\star$

$$\mathrm{pk} = \mathrm{sk} \star E$$

CSIDH

Start with a prime $p = 4\ell_1 \cdot \ell_n - 1$ with ℓ_i small primes.

CSIDH

Start with a prime $p = 4\ell_1 \cdot \ell_n - 1$ with ℓ_i small primes.

There is a abelian group G acting on a set of elliptic curves $\mathcal{E} = \{E/\mathbb{F}_p : \#E(\mathbb{F}_p) = p + 1\}$, represented in Montgomery form

$$E_A : y^2 = x^3 + Ax^2 + x \quad \text{for some } A \in \mathbb{F}_p^* \setminus \{\pm 2\}$$

CSIDH

Start with a prime $p = 4\ell_1 \cdot \ell_n - 1$ with ℓ_i small primes.

There is a abelian group G acting on a set of elliptic curves $\mathcal{E} = \{E/\mathbb{F}_p : \#E(\mathbb{F}_p) = p + 1\}$, represented in Montgomery form

$$E_A : y^2 = x^3 + Ax^2 + x \quad \text{for some } A \in \mathbb{F}_p^* \setminus \{\pm 2\}$$

For every $\ell_i \mid p + 1$, we have a group element $g_i \in G$ with efficient action via isogenies:

$$E_{A'} = g_i \star E_A. \qquad \longleftrightarrow \qquad \phi : E_A \to E_{A'} \quad \ell_i\text{-isogeny.}$$

CSIDH

Start with a prime $p = 4\ell_1 \cdot \ell_n - 1$ with ℓ_i small primes.
There is a abelian group G acting on a set of elliptic curves $\mathcal{E} = \{E/\mathbb{F}_p : \#E(\mathbb{F}_p) = p + 1\}$, represented in Montgomery form

$$E_A : y^2 = x^3 + Ax^2 + x \quad \text{for some } A \in \mathbb{F}_p^* \setminus \{\pm 2\}$$

For every $\ell_i \mid p + 1$, we have a group element $g_i \in G$ with efficient action via isogenies:

$$E_{A'} = g_i \star E_A. \qquad \longleftrightarrow \qquad \phi : E_A \to E_{A'} \quad \ell_i\text{-isogeny.}$$

Secret keys $(e_1, \ldots, e_n) \in \mathbb{Z}^n$; public keys

$$E_{A'} = \left(\prod_{i=1}^{n} g_i^{e_i} \right) \star E_A.$$

STEP-BY-STEP PROCEDURE

- **Identify the curve**: Start with a supersingular elliptic curve E over $\mathbb{F}_p$.
- **Select a point**: Choose a point P of order 3 on E.
- **Compute the kernel polynomial**: The kernel polynomial $K_P(x)$ is computed using the x-coordinates of P.
- **Evaluate the isogeny**: Construct the isogeny $\phi : E \to E'$ with kernel generated by P.
- **Iterate using the secret exponent**: Apply the isogeny e_3 times to compute the final curve.

CSIDH - COMPUTATION

Step 1: Identify the Curve

CSIDH - COMPUTATION

Step 1: Identify the Curve

- Given a supersingular elliptic curve E defined over $\mathbb{F}_p$

CSIDH - COMPUTATION

Step 1: Identify the Curve

- Given a supersingular elliptic curve E defined over $\mathbb{F}_p$
- Example: $E : y^2 = x^3 + Ax + B$ with specific A and B
-
-
-

CSIDH - COMPUTATION

Step 1: Identify the Curve

- Given a supersingular elliptic curve E defined over $\mathbb{F}_p$
- Example: $E : y^2 = x^3 + Ax + B$ with specific A and B

Step 2: Select a Point

CSIDH - COMPUTATION

Step 1: Identify the Curve

- Given a supersingular elliptic curve E defined over $\mathbb{F}_p$
- Example: $E : y^2 = x^3 + Ax + B$ with specific A and B

Step 2: Select a Point

- Choose a point P on E of order 3

CSIDH - COMPUTATION

Step 1: Identify the Curve

- Given a supersingular elliptic curve E defined over $\mathbb{F}_p$
- Example: $E : y^2 = x^3 + Ax + B$ with specific A and B

Step 2: Select a Point

- Choose a point P on E of order 3
- Ensure P is not the point at infinity
-

CSIDH - COMPUTATION

Step 1: Identify the Curve

- Given a supersingular elliptic curve E defined over $\mathbb{F}_p$
- Example: $E : y^2 = x^3 + Ax + B$ with specific A and B

Step 2: Select a Point

- Choose a point P on E of order 3
- Ensure P is not the point at infinity
- Example: $P = (x_1, y_1)$

CSIDH - COMPUTATION

Step 3: Compute the Kernel Polynomial

CSIDH - COMPUTATION

Step 3: Compute the Kernel Polynomial

- Kernel polynomial $K_P(x)$ is given by $(x - x(P))$

CSIDH - COMPUTATION

Step 3: Compute the Kernel Polynomial

- Kernel polynomial $K_P(x)$ is given by $(x - x(P))$
- For a point $P = (x_1, y_1)$, $K_P(x) = x - x_1$

CSIDH - COMPUTATION

Step 3: Compute the Kernel Polynomial

- Kernel polynomial $K_P(x)$ is given by $(x - x(P))$
- For a point $P = (x_1, y_1)$, $K_P(x) = x - x_1$

Step 4: Evaluate the Isogeny

CSIDH - COMPUTATION

Step 3: Compute the Kernel Polynomial

- Kernel polynomial $K_P(x)$ is given by $(x - x(P))$
- For a point $P = (x_1, y_1)$, $K_P(x) = x - x_1$

Step 4: Evaluate the Isogeny

- Construct the isogeny ϕ with kernel $\langle P \rangle$
-
-

CSIDH - COMPUTATION

Step 3: Compute the Kernel Polynomial

- Kernel polynomial $K_P(x)$ is given by $(x - x(P))$
- For a point $P = (x_1, y_1)$, $K_P(x) = x - x_1$

Step 4: Evaluate the Isogeny

- Construct the isogeny ϕ with kernel $\langle P \rangle$
- Use Vélu's formulas to compute the new curve coefficients
-

CSIDH - COMPUTATION

Step 3: Compute the Kernel Polynomial

- Kernel polynomial $K_P(x)$ is given by $(x - x(P))$
- For a point $P = (x_1, y_1)$, $K_P(x) = x - x_1$

Step 4: Evaluate the Isogeny

- Construct the isogeny ϕ with kernel $\langle P \rangle$
- Use Vélu's formulas to compute the new curve coefficients
- Vélu's formulas for ϕ:

$$\phi(x) = x - \sum_{R \in \langle P \rangle \setminus \{O\}} \left(\frac{x - x(R)}{x - x(P + R)} \right)$$

$$\phi(y) = y \prod_{R \in \langle P \rangle \setminus \{O\}} \left(\frac{x - x(R)}{x - x(P + R)} \right)^{1/2}$$

Step 5: Iterate using the Secret Exponent

Step 5: Iterate using the Secret Exponent

- Let e be the secret exponent;
-
-

Step 5: Iterate using the Secret Exponent

- Let e be the secret exponent;
- Apply the degree-3 isogeny ϕ to E repeatedly e times;
-

Step 5: Iterate using the Secret Exponent

- Let e be the secret exponent;
- Apply the degree-3 isogeny ϕ to E repeatedly e times;
- After e applications, compute the final curve E'.

CONSTANT-TIME EVALUATION

Constant-time evaluation of the group action If the input is a CSIDH curve and a private key, and the output is the result of the CSIDH action, then the algorithm time provides no information about the private key, and provides no information about the output.

Secret keys $(e_1, \ldots, e_n) \in \mathbb{Z}^n$; public keys

$$E_{A'} = \left(\prod_{i=1}^{n} g_i^{e_i}\right) \star E_A.$$

BATCHING

The batching idea

CSIDH−512 prime $p = 4 \cdot (3 \cdot 5 \cdot \cdots \cdot 373 \cdot 587) - 1$.

BATCHING

The batching idea

`CSIDH-512` prime $p = 4 \cdot (3 \cdot 5 \cdot \cdots \cdot 373 \cdot 587) - 1$.

We start with the exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$:

primes	3	5	7	11	13	17	19	23	29	31	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...

The batching idea

CSIDH−512 prime $p = 4 \cdot (3 \cdot 5 \cdot \cdots \cdot 373 \cdot 587) - 1$.

We start with the exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$.

Now we split the primes into batches:

primes	{ 3	5	7 }	{ 11	13	17	19 }	{ 23	29	31 }	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...

BATCHING

The batching idea

`CSIDH-512` prime $p = 4 \cdot (3 \cdot 5 \cdot \cdots \cdot 373 \cdot 587) - 1$.

We start with the exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$.

Now we group the entries in the exponent vector isogenies per batch:

primes	{ 3	5	7 }	{ 11	13	17	19 }	{ 23	29	31 }	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...
per batch		3			5				3		

exponent vector $(e_1, \ldots, e_n) \in \mathbb{Z}^n$ comes from the subset in which we compute

- 3 {3, 5, 7}-isogenies,
- 5 {11, 13, 17, 19}-isogenies,
- and 3 {23, 29, 31}-isogenies.

The batching idea

`CSIDH−512` prime $p = 4 \cdot (3 \cdot 5 \cdot \dots \cdot 373 \cdot 587) - 1$.

We start with the exponent vector $(e_1, \dots, e_n) \in \mathbb{Z}^n$.

Now we group the entries in the exponent vector isogenies per batch:

primes	{3	5	7}	{11	13	17	19}	{23	29	31}	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...
per batch		3			5				3		

exponent vector $(e_1, \dots, e_n) \in \mathbb{Z}^n$ comes from the subset in which we compute

- up to 3 {3, 5, 7}-isogenies,
- up to 5 {11, 13, 17, 19}-isogenies,
- and up to 3 {23, 29, 31}-isogenies.

The batching idea

CSIDH−512 prime $p = 4 \cdot (3 \cdot 5 \cdot \cdots \cdot 373 \cdot 587) - 1$.

Now we group the isogenies per batch:

primes	{ 3	5	7 }	{ 11	13	17	19 }	{ 23	29	31 }	...
exponent vector	1	-2	0	3	-1	1	0	2	-1	0	...
per batch		3			5				3		

Batching Keyspace For B batches: For $N \in \mathbb{Z}^B_{>0}$ and $m \in \mathbb{Z}^B_{\geq 0}$, we define

$$\mathcal{K}_{N,m} := \left\{(e_1, \ldots, e_n) \in \mathbb{Z}^n \mid \sum_{j=1}^{N_i} |e_{i,j}| \leq m_i \text{ for } 1 \leq i \leq B\right\}.$$

ISOGENY MAGIC

In CSIDH, start with prime $p = 4\ell_1 \ldots \ell_n - 1$ for ℓ_i small odd primes. Group action For every $\ell_i \mid p + 1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Group action via isogenies

In CSIDH, start with prime $p = 4\ell_1 \ldots \ell_n - 1$ for ℓ_i small odd primes. Group action For every $\ell_i \mid p + 1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Group action via isogenies

Replace the group element g_i

$$g_i : E_A \mapsto E_{A'}$$

ISOGENY MAGIC

In CSIDH, start with prime $p = 4\ell_1 \ldots \ell_n - 1$ for ℓ_i small odd primes. Group action For every $\ell_i \mid p + 1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Group action via isogenies

Replace the group element g_i with an ℓ_i-isogeny ϕ:

$$\phi : E_A \to E_{A'}$$

Isogenies are algebraic group homomorphisms of elliptic curves

$$\phi : y^2 = x^3 + Ax^2 + x \longrightarrow y^2 = x^3 + A'x^2 + x$$

$$(x, y) \longmapsto (f(x, y), g(x, y))$$

f, g rational functions over $\mathbb{F}_p$

ISOGENY MAGIC

In CSIDH, start with prime $p = 4\ell_1 \ldots \ell_n - 1$ for ℓ_i small odd primes. Group action For every $\ell_i \mid p+1$, we have an element g_i that we can act with using ℓ_i-isogenies:

$$E_{A'} = g_i \star E_A$$

Group action via isogenies

Replace the group element g_i with an ℓ_i-isogeny ϕ:

$$\phi : E_A \to E_{A'}$$

Isogenies are algebraic group homomorphisms of elliptic curves:

$$P \in E_A \longmapsto \phi(P) \in E_{A'}$$

$$\text{order } \ell_i N \longrightarrow \text{order } N.$$

COMPUTING THE GROUP ACTION

Computing the action by $g_i \leftrightarrow \ell_i$ Simplified algorithm to compute the group action $E_{A'} = g_i \star E_A$:

1. find a point P of order ℓ_i on E_A:
   1. generate a point T of order $p+1$ on E_A,
   2. multiply $P = [\frac{p+1}{\ell_i}]T$.
2. Compute the ℓ_i-isogeny $\phi : E_A \rightarrow E_{A'}$ with kernel P:
   1. enumerate the multiples $[i]P$ of the point P for $i \in S$, with $S = \{1, 2, \ldots, \frac{\ell-1}{2}\}$ [**?**] or $S = \{1, 3, 5, \ldots, \ell - 2\}$ [**?**],
   2. construct a polynomial $h(X) = \prod_{i \in S}(x - x([i]P))$,
   3. Compute the coefficient A' from $h(X)$.

COMPUTING THE GROUP ACTION

Computing the action by $g_i \leftrightarrow \ell_i$ Simplified algorithm to compute the group action $E_{A'} = g_i \star E_A$:

1. find a point P of order ℓ_i on E_A:
   1. generate a point T of order $p+1$ on E_A,
   2. multiply $P = [\frac{p+1}{\ell_i}]T$.
2. Compute the ℓ_i-isogeny $\phi : E_A \to E_{A'}$ with kernel P:
   1. enumerate the multiples $[i]P$ of the point P for $i \in S$, with $S = \{1, 2, \ldots, \frac{\ell-1}{2}\}$ [**?**] or $S = \{1, 3, 5, \ldots, \ell - 2\}$ [**?**],
   2. construct a polynomial $h(X) = \prod_{i \in S}(x - x([i]P))$,
   3. Compute the coefficient A' from $h(X)$.

COMPUTING THE GROUP ACTION

Computing the action by $g_i \leftrightarrow \ell_i$ Simplified algorithm to compute the group action $E_{A'} = g_i \star E_A$:

1. find a point P of order ℓ_i on E_A:
   1. generate a point T of order $p+1$ on E_A,
   2. multiply $P = [\frac{p+1}{\ell_i}]T$. Costs $\approx 10\log_2(p)$ mult in $\mathbb{F}_p$.
2. Compute the ℓ_i-isogeny $\phi : E_A \to E_{A'}$ with kernel P: Cost $\leq 6\ell_i$ mult in $\mathbb{F}_p$
   1. enumerate the multiples $[i]P$ of the point P for $i \in S$, with $S = \{1, 2, \dots, \frac{\ell-1}{2}\}$ [**?**] or $S = \{1, 3, 5, \dots, \ell-2\}$ [**?**],
   2. construct a polynomial $h(X) = \prod_{i \in S}(x - x([i]P))$,
   3. Compute the coefficient A' from $h(X)$.

AMORTIZE THE COST

Exponent vector $(1, 1, 1, 0, \ldots, 0)$

We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_2 = 5$ and $\ell_3 = 7$:

1. Find a suitable point:
   1. Generate a random point T of order $p + 1$,
   2. Compute $T_1 = \left[\frac{p+1}{3 \cdot 5 \cdot 7}\right] T$ has exact order $3 \cdot 5 \cdot 7$,
2. Compute the isogenies:
   1. 3-isogeny:
      1. Compute $P_1 = [5 \cdot 7]T_1$ has order 3,
      2. Use P_1 to construct 3-isogeny ϕ_1,
      3. Point $T_2 = \phi_1(T_1)$ has order $5 \cdot 7$ on the new curve,
   2. 5-isogeny:
      1. Compute $P_2 = [7]T_2$ has order 5,
      2. Construct 5-isogeny ϕ_2 with kernel P_2,

AMORTIZE THE COST

Exponent vector $(1, 1, 1, 0, \dots, 0)$

We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_2 = 5$ and $\ell_3 = 7$:

1. Find a suitable point:
    1. Generate a random point T of order $p + 1$,
    2. Compute $T_1 = \left[\frac{p+1}{3\cdot5\cdot7}\right] T$ has exact order $3 \cdot 5 \cdot 7$,
2. Compute the isogenies:
    1. 3-isogeny:
        1. Compute $P_1 = [5 \cdot 7]T_1$ has order 3,
        2. Use P_1 to construct 3-isogeny ϕ_1,
        3. Point $T_2 = \phi_1(T_1)$ has order $5 \cdot 7$ on the new curve,
    2. 5-isogeny:
        1. Compute $P_2 = [7]T_2$ has order 5,
        2. Construct 5-isogeny ϕ_2 with kernel P_2,

TOWARDS ATOMIC BLOCKS

Exponent vector $(1, 0, 1, 0, \ldots, 0)$ We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_3 = 7$ but no 5-isogeny:

1. Find a suitable point:
   1. Generate a random point T of order $p + 1$,
   2. Compute $T_1 = \left[\frac{p+1}{3 \cdot 5 \cdot 7}\right] T$ has exact order $3 \cdot 5 \cdot 7$,
2. Compute the isogenies:
   1. 3-isogeny:
      1. Compute $P_1 = [5 \cdot 7]T_1$ has order 3,
      2. Use P_1 to construct 3-isogeny ϕ_1,
      3. Point $T_2 = \phi_1(T_1)$ has order $5 \cdot 7$ on the new curve,
   2. No 5-isogeny:
      1. Compute the isogeny as before but throw away the results,
      2. Adjust to code to always compute $[5]T_2$,
      3. The point $T_3 = [5]T_2$ has order 7 on the same curve,

TOWARDS ATOMIC BLOCKS

Exponent vector $(1, 0, 1, 0, \ldots, 0)$ We compute ℓ_i-isogenies for $\ell_1 = 3$ and $\ell_3 = 7$ but no 5-isogeny:

1. Find a suitable point:
   1. Generate a random point T of order $p + 1$,
   2. Compute $T_1 = \left[\frac{p+1}{3 \cdot 5 \cdot 7}\right] T$ has exact order $3 \cdot 5 \cdot 7$,
2. Compute the isogenies:
   1. 3-isogeny:
      1. Compute $P_1 = [5 \cdot 7]T_1$ has order 3,
      2. Use P_1 to construct 3-isogeny ϕ_1,
      3. Point $T_2 = \phi_1(T_1)$ has order $5 \cdot 7$ on the new curve,
   2. No 5-isogeny:
      1. Compute the isogeny as before but throw away the results,
      2. Adjust to code to always compute $[5]T_2$,
      3. The point $T_3 = [5]T_2$ has order 7 on the same curve,

ATOMIC BLOCKS

Definition (Atomic Blocks, simplified)

Let $I = (I_1, \ldots, I_k) \in \mathbb{Z}^k$ be such that $1 \le I_1 < I_2 < \cdots < I_k \le n$.

An *atomic block* of length k is a probabilistic algorithm $_I$ taking inputs A and $\epsilon \in \{0,1\}^k$ and returning $A' \in \mathbb{F}_p$ such that $E_{A'} = (\prod_i g_{I_i}^{\epsilon_i}) \star E_A$, satisfying

> there is a function τ such that, for each (A, ϵ) the distribution of the time taken by $_I$, given that A' is returned by $_I$ on input (A, ϵ), is $\tau(I)$.

Evaluating 3, 5, and 7-isogeny

On the previous slide, we saw an atomic block $_I$ with $I = (1, 2, 3)$ that computes

$$E_{A'} = g_1^{\epsilon_1} g_2^{\epsilon_2} g_3^{\epsilon_3} \star E_A$$

for $(\epsilon_1, \epsilon_2, \epsilon_3) \in \{0,1\}^3$ without leaking timing information about $(\epsilon_1, \epsilon_2, \epsilon_3)$.

Definition (Atomic Blocks, simplified)

Let $I = (I_1, \ldots, I_k) \in \mathbb{Z}^k$ be such that $1 \leq I_1 < I_2 < \cdots < I_k \leq n$.

An *atomic block* of length k is a probabilistic algorithm $_I$ taking inputs A and $\epsilon \in \{0,1\}^k$ and returning $A' \in \mathbb{F}_p$ such that $E_{A'} = (\prod_i g_{I_i}^{\epsilon_i}) \star E_A$, satisfying

> there is a function τ such that, for each (A, ϵ) the distribution of the time taken by $_I$, given that A' is returned by $_I$ on input (A, ϵ), is $\tau(I)$.

Evaluating 3, 5, and 7-isogeny

On the previous slide, we saw an atomic block $_I$ with $I = (1, 2, 3)$ that computes

$$E_{A'} = g_1^{\epsilon_1} g_2^{\epsilon_2} g_3^{\epsilon_3} \star E_A$$

for $(\epsilon_1, \epsilon_2, \epsilon_3) \in \{0,1\}^3$ without leaking timing information about $(\epsilon_1, \epsilon_2, \epsilon_3)$.

ATOMIC BLOCKS FOR BATCHES

Atomic blocks for batches

Suppose we have batches $\{3, 5, 7\}, \{11, 13, 17\}, \ldots$ And we want to compute one 5-isogeny and one 11-isogeny, i.e. exponent vector $(0, 1, 0, 1, 0, 0, 0, \ldots)$

1. Find a suitable point:
   1. Generate a random point T of order $p + 1$,
   2. Compute $T_1 = \left[\frac{p+1}{(3\cdot5\cdot7)(11\cdot13\cdot17)}\right] T$ has order $(3 \cdot 5 \cdot 7)(11 \cdot 13 \cdot 17)$.
2. Compute the isogenies:
   1. $\{3, 5, 7\}$-isogeny:
      1. Compute $P_1 = [(11 \cdot 13 \cdot 17)]T_1$ has order $(3 \cdot 5 \cdot 7)$,
      2. Use $[3 \cdot 7]P_1$ of order 5 to construct 5 -isogeny ϕ_1,
      3. Point $T_2 = [3 \cdot 7]\phi_1(T_1)$ has order $11 \cdot 13 \cdot 17$ on the new curve,
   2. $\{11, 13, 17\}$-isogeny:
      1. Compute $P_2 = [13 \cdot 17]T_2$ has order 11,
      2. Construct 11-isogeny ϕ_2 with kernel P_2

ATOMIC BLOCKS FOR BATCHES

Atomic blocks for batches

Suppose we have batches $\{3, 5, 7\}, \{11, 13, 17\}, \ldots$ And we want to compute one 5-isogeny and one 11-isogeny, i.e. exponent vector $(0, 1, 0, 1, 0, 0, 0, \ldots)$

1. Find a suitable point:
    1. Generate a random point T of order $p + 1$,
    2. Compute $T_1 = \left[\frac{p+1}{(3\cdot5\cdot7)(11\cdot13\cdot17)}\right] T$ has order $(3 \cdot 5 \cdot 7)(11 \cdot 13 \cdot 17)$.
2. Compute the isogenies:
    1. $\{3, 5, 7\}$-isogeny:
        1. Compute $P_1 = [(11 \cdot 13 \cdot 17)]T_1$ has order $(3 \cdot 5 \cdot 7)$,
        2. Use $[3 \cdot 7]P_1$ of order 5 to construct 5 -isogeny ϕ_1,
        3. Point $T_2 = [3 \cdot 7]\phi_1(T_1)$ has order $11 \cdot 13 \cdot 17$ on the new curve,
    2. $\{11, 13, 17\}$-isogeny:
        1. Compute $P_2 = [13 \cdot 17]T_2$ has order 11,
        2. Construct 11-isogeny ϕ_2 with kernel P_2.

ATOMIC BLOCKS FOR BATCHES

Suppose we have batches $\{3, 5, 7\}, \{11, 13, 17\}, \ldots$ And we want to compute one 5-isogeny and one 11-isogeny, i.e. exponent vector $(0, 1, 0, 1, 0, 0, 0, \ldots)$

1. Find a suitable point:
    1. Generate a random point T of order $p + 1$,
    2. Compute $T_1 = \left[\frac{p+1}{(3 \cdot 5 \cdot 7)(11 \cdot 13 \cdot 17)}\right] T$ has order $(3 \cdot 5 \cdot 7)(11 \cdot 13 \cdot 17)$.
2. Compute the isogenies:
    1. $\{3, 5, 7\}$-isogeny:
        1. Compute $P_1 = [(11 \cdot 13 \cdot 17)]T_1$ has order $(3 \cdot 5 \cdot 7)$,
        2. Use $[3 \cdot 7]P_1$ of order 5 to construct 5-isogeny ϕ_1,
        3. Point $T_2 = [3 \cdot 7]\phi_1(T_1)$ has order $11 \cdot 13 \cdot 17$ on the new curve,
    2. $\{11, 13, 17\}$-isogeny:
        1. Compute $P_2 = [13 \cdot 17]T_2$ has order 11,
        2. Construct 11-isogeny ϕ_2 with kernel P_2.

MATRYOSHKA ISOGENY

How to construct the isogeny with the same code for all primes in the batch:

MATRYOSHKA ISOGENY

How to construct the isogeny with the same code for all primes in the batch:

Matryoshka Isogeny for the batch $\{11, 13, 17\}$

Compute the 11-isogeny

1. enumerate the multiples $[i]P$ of the point P for $i \in S$,

   with $S = \{1, 2, \ldots, 5\}$

2. construct $h(X) = \prod_{i=1}^{5}(x - x([i]P))$,

3. Compute the coefficient A' from $h(X)$.

MATRYOSHKA ISOGENY

How to construct the isogeny with the same code for all primes in the batch:

Matryoshka Isogeny for the batch $\{11, 13, 17\}$

Compute the $\cancel{11}$ 13-isogeny

1 enumerate the multiples $[i]P$ of the point P for $i \in S$,

with $S = \{1, 2, \ldots, 5, 6\}$

2 construct $h(X) = \prod_{i=1}^{5}(x - x([i]P)) \cdot (x - x([6]P))$,

3 Compute the coefficient A' from $h(X)$.

MATRYOSHKA ISOGENY

How to construct the isogeny with the same code for all primes in the batch:

Matryoshka Isogeny for the batch $\{11, 13, 17\}$

Compute the $\cancel{11}$ $\cancel{13}$ 17-isogeny

1. enumerate the multiples $[i]P$ of the point P for $i \in S$,

   with $S = \{1, 2, \ldots, 5, 6, 7, 8\}$

2. construct $h(X) = \prod_{i=1}^{5}(x - x([i]P)) \cdot (x - x([6]P)) \cdot (x - x([7]P))(x - x([8]P))$,

3. Compute the coefficient A' from $h(X)$.

QUICK RECAP

- We need to evaluate where the code will ran;
- We should use tools to verify our constant-time implementation;
- We might need to adapt a scheme to adequate constant-time;
- We have other ways of secure implement an implementation.

OPEN PROBLEMS

- Isogenies are a candidate on NIST (SQISign);
- Falcon needs a secure implementation (Floating point arithmetic);
- Improvements on the hardware implementation of Lattice schemes;
- Secure implementations of code-base schemes (HQC, McEliece, Bike).

QUESTIONS

Thank you to listen.

Questions?

gustavo@crytpme.in

Is This Constant Time?

A CHES drinking game

Game

- We'll show C/C++ code snippets along with compiler (including version), compilation flags and targets

- Guess if the code is constant time or not

- **Don't use Godbolt.org or you'll kill the fun ☺**

- **If the majority of the room gets it right, we drink**

ARM modular arithmetic

ARMv7-a Clang 11.0, -Ofast -mtune=cortex-m4

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint64_t a) {
    return a % Q;
}
```

ARM modular arithmetic

ARMv7-a Clang 11.0, -Ofast -mtune=cortex-m4

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint64_t a) {
    return a % Q;
}
```

```
mod_q(unsigned long long):
        push    {r11, lr}
        ldr     r2, .LCPI0_0
        mov     r3, #0
        bl      __aeabi_uldivmod
        mov     r0, r2
        pop     {r11, lr}
        bx      lr
.LCPI0_0:
        .long   8380417
```

ARM modular arithmetic

ARMv7-a Clang 11.0, -Ofast -mtune=cortex-m4

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint64_t a) {
    return a % Q;
}
```

```asm
d_q(unsigned long long):
        push    {r11, lr}
        ldr     r2, .LCPI0_0
        mov     r3, #0
        bl      __aeabi_uldivmod
        mov     r0, r2
        pop     {r11, lr}
        bx      lr
.LCPI0_0:
        .long   8380417
```

modular arithmetic

ARMv8-a Clang 16.0, -Ofast -mtune=cortex-m4

```
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint32_t a){
    return a % Q;
}
```

modular arithmetic

ARMv8-a Clang 16.0, -Ofast -mtune=cortex-m4

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint32_t a){
    return a % Q;
}
```

```asm
mod_q(unsigned int):
        ldr     r1, .LCPI0_0
        umull   r2, r3, r0, r1
        ldr     r2, .LCPI0_1
        lsr     r1, r3, #21
        mul     r3, r1, r2
        sub     r0, r0, r3
        bx      lr
.LCPI0_0:
        .long   1074791297
.LCPI0_1:
        .long   8380417
```

modular arithmetic

ARMv8-a Clang 16.0, -Ofast -mtune=cortex-m4

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint32_t a){
    return a % Q;
}
```

```asm
mod_q(unsigned int):
        ldr     r1, .LCPI0_0
        umull   r2, r3, r0, r1
        ldr     r2, .LCPI0_1
        lsr     r1, r3, #21
        mul     r3, r1, r2
        sub     r0, r0, r3
        bx      lr
.LCPI0_0:
        .long   1074791297
.LCPI0_1:
        .long   8380417
```

modular arithmetic

ARMv7-a Clang 16.0, –Ofast -mtune=cortex-m3

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint32_t a){
    return a % Q;
}
```

modular arithmetic

ARMv7-a Clang 16.0, –Ofast -mtune=cortex-m3

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint32_t a){
    return a % Q;
}
```

```
mod_q:
    ldr r1, .LCPI0_0
    umull r2, r3, r0, r1
    ldr r2, .LCPI0_1
    lsr r1, r3, #21
    mul r3, r1, r2
    sub r0, r0, r3
    bx lr
.LCPI0_0:
    .long 1074791297 @ 0x40100381
.LCPI0_1:
    .long 8380417 @ 0x7fe001
```

modular arithmetic

Compact Dilithium implementations on Cortex-M3 and Cortex M-4
Greconici et Al.

ARMv7-a Clang 16.0, –Ofast -mtune=c… m3

```c
#include <stdint.h>

#define Q 8380417

uint32_t mod_q(uint32_t a){
    return a % Q;
}
```

```asm
mod_q:
    ldr r1, .LCPI0_0
    umull r2, r3, r0, r1
    ldr r2, .LCPI0_1
    lsr r1, r3, #21
    mul r3, r1, r2
    sub r0, r0, r3
    bx lr
.LCPI0_0:
    .long 1074791297 @ 0x40100381
.LCPI0_1:
    .long 8380417 @ 0x7fe001
```

Long unsigned	`UMULL RdLo, RdHi, Rn, Rm`	3 to 5[a]

HQC reference code

NOT CLAIMED TO BE CONSTANT-TIME

X86-64 Clang-16.0, -Ofast

```
#include <stdint.h>
typedef int16_t expandedCodeword[128];

int32_t find_peaks(expandedCodeword *transform) {
    int32_t peak_abs_value = 0;
    int32_t peak_value = 0;
    int32_t peak_pos = 0;
    for (int32_t i = 0; i < 128; i++) {
        // get absolute value
        int32_t t = (*transform)[i];
        int32_t absolute = t < 0 ? -t : t;

        // int32_t pos_mask = -(t > 0);
        // all compilers nowadays compile with a conditional move
        peak_value = absolute > peak_abs_value ? t : peak_value;
        peak_pos = absolute > peak_abs_value ? i : peak_pos;
        peak_abs_value = absolute > peak_abs_value ? absolute : peak_abs_value;
    }
    // set bit 7
    peak_pos |= 128 * (peak_value > 0);
    return peak_pos;
}
```

HQC reference code

NOT CLAIMED TO BE CONSTANT-TIME

X86-64 Clang-16.0, -Ofast

```c
#include <stdint.h>
typedef int16_t expandedCodeword[128];

int32_t find_peaks(expandedCodeword *transform) {
    int32_t peak_abs_value = 0;
    int32_t peak_value = 0;
    int32_t peak_pos = 0;
    for (int32_t i = 0; i < 128; i++) {
        // get absolute value
        int32_t t = (*transform)[i];
        int32_t absolute = t < 0 ? -t : t;

        // int32_t pos_mask = -(t > 0);
        // all compilers nowadays compile with a conditional move
        peak_value = absolute > peak_abs_value ? t : peak_value;
        peak_pos = absolute > peak_abs_value ? i : peak_pos;
        peak_abs_value = absolute > peak_abs_value ? absolute : peak_abs_value;
    }
    // set bit 7
    peak_pos |= 128 * (peak_value > 0);
    return peak_pos;
}
```

```asm
find_peaks:                             # @find_peaks
        xor     eax, eax
        xor     edx, edx
        xor     ecx, ecx
        xor     esi, esi
.LBB0_1:                                # =>This Inner Loop Header: Depth=1
        movsx   r8d, word ptr [rdi + 2*rax]
        mov     r9d, r8d
        neg     r9d
        cmovs   r9d, r8d
        cmp     r9d, edx
        cmovg   ecx, r8d
        mov     r8d, esi
        cmovg   r8d, eax
        cmovle  r9d, edx
        movsx   esi, word ptr [rdi + 2*rax + 2]
        mov     edx, esi
        neg     edx
        cmovs   edx, esi
        cmp     edx, r9d
        cmova   ecx, esi
        lea     esi, [rax + 1]
        cmovbe  esi, r8d
        cmovle  edx, r9d
        add     rax, 2
        cmp     rax, 128
        jne     .LBB0_1
        xor     eax, eax
        test    ecx, ecx
        setg    al
        shl     eax, 7
        or      eax, esi
        ret
```

HQC reference code

NOT CLAIMED TO BE CONSTANT-TIME

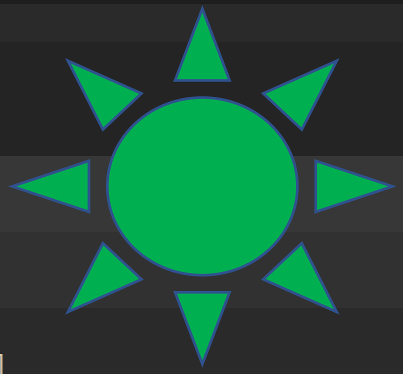
X86-64 Clang-16.0, -Ofast

```c
#include <stdint.h>
typedef int16_t expandedCodeword[128];

int32_t find_peaks(expandedCodeword *transform) {
    int32_t peak_abs_value = 0;
    int32_t peak_value = 0;
    int32_t peak_pos = 0;
    for (int32_t i = 0; i < 128; i++) {
        // get absolute value
        int32_t t = (*transform)[i];
        int32_t absolute = t < 0 ? -t : t;

        // int32_t pos_mask = -(t > 0);
        // all compilers nowadays compile with a conditional move
        peak_value = absolute > peak_abs_value ? t : peak_value;
        peak_pos = absolute > peak_abs_value ? i : peak_pos;
        peak_abs_value = absolute > peak_abs_value ? absolute : peak_abs_value;
    }
    // set bit 7
    peak_pos |= 128 * (peak_value > 0);
    return peak_pos;
}
```

```asm
find_peaks:                             # @find_peaks
        xor     eax, eax
        xor     edx, edx
        xor     ecx, ecx
        xor     esi, esi
.LBB0_1:                                # =>This Inner Loop Header: Depth=1
        movsx   r8d, word ptr [rdi + 2*rax]
        mov     r9d, r8d
        neg     r9d
        cmovs   r9d, r8d
        cmp     r9d, edx
        cmovg   ecx, r8d
        mov     r8d, esi
        cmovg   r8d, eax
        cmovle  r9d, edx
        movsx   esi, word ptr [rdi + 2*rax + 2]
        mov     edx, esi
        neg     edx
        cmovs   edx, esi
        cmp     edx, r9d
        cmova   ecx, esi
        lea     esi, [rax + 1]
        cmovbe  esi, r8d
        cmovle  edx, r9d
        add     rax, 2
        cmp     rax, 128
        jne     .LBB0_1
        xor     eax, eax
        test    ecx, ecx
        setg    al
        shl     eax, 7
        or      eax, esi
        ret
```



HQC reference code

NOT CLAIMED TO BE CONSTANT-TIME

X86-64 GCC 13.2, -Ofast

```c
#include <stdint.h>
typedef int16_t expandedCodeword[128];

int32_t find_peaks(expandedCodeword *transform) {
    int32_t peak_abs_value = 0;
    int32_t peak_value = 0;
    int32_t peak_pos = 0;
    for (int32_t i = 0; i < 128; i++) {
        // get absolute value
        int32_t t = (*transform)[i];
        int32_t absolute = t < 0 ? -t : t;

        // int32_t pos_mask = -(t > 0);
        // all compilers nowadays compile with a conditional move
        peak_value = absolute > peak_abs_value ? t : peak_value;
        peak_pos = absolute > peak_abs_value ? i : peak_pos;
        peak_abs_value = absolute > peak_abs_value ? absolute : peak_abs_value;
    }
    // set bit 7
    peak_pos |= 128 * (peak_value > 0);
    return peak_pos;
}
```

HQC reference code

NOT CLAIMED TO BE CONSTANT-TIME

X86-64 GCC 13.2, -Ofast

```c
#include <stdint.h>
typedef int16_t expandedCodeword[128];

int32_t find_peaks(expandedCodeword *transform) {
    int32_t peak_abs_value = 0;
    int32_t peak_value = 0;
    int32_t peak_pos = 0;
    for (int32_t i = 0; i < 128; i++) {
        // get absolute value
        int32_t t = (*transform)[i];
        int32_t absolute = t < 0 ? -t : t;

        // int32_t pos_mask = -(t > 0);
        // all compilers nowadays compile with a conditional move
        peak_value = absolute > peak_abs_value ? t : peak_value;
        peak_pos = absolute > peak_abs_value ? i : peak_pos;
        peak_abs_value = absolute > peak_abs_value ? absolute : peak_abs_value;
    }
    // set bit 7
    peak_pos |= 128 * (peak_value > 0);
    return peak_pos;
}
```

```asm
find_peaks:
        xor     edx, edx
        xor     r8d, r8d
        xor     r9d, r9d
        xor     esi, esi
.L3:
        movzx   ecx, WORD PTR [rdi+rdx*2]
        mov     eax, ecx
        neg     ax
        cmovs   eax, ecx
        movzx   eax, ax
        cmp     eax, esi
        jle     .L2
        mov     r8d, edx
        movsx   r9d, cx
.L2:
        cmp     esi, eax
        cmovl   esi, eax
        add     rdx, 1
        cmp     rdx, 128
        jne     .L3
        xor     eax, eax
        test    r9d, r9d
        setg    al
        sal     eax, 7
        or      eax, r8d
        ret
```

HQC reference code

NOT CLAIMED TO BE CONSTANT-TIME

X86-64 GCC 13.2, -Ofast

```c
#include <stdint.h>
typedef int16_t expandedCodeword[128];

int32_t find_peaks(expandedCodeword *transform) {
    int32_t peak_abs_value = 0;
    int32_t peak_value = 0;
    int32_t peak_pos = 0;
    for (int32_t i = 0; i < 128; i++) {
        // get absolute value
        int32_t t = (*transform)[i];
        int32_t absolute = t < 0 ? -t : t;

        // int32_t pos_mask = -(t > 0);
        // all compilers nowadays compile with a conditional move
        peak_value = absolute > peak_abs_value ? t : peak_value;
        peak_pos = absolute > peak_abs_value ? i : peak_pos;
        peak_abs_value = absolute > peak_abs_value ? absolute : peak_abs_value;
    }
    // set bit 7
    peak_pos |= 128 * (peak_value > 0);
    return peak_pos;
}
```

```asm
find_peaks:
        xor     edx, edx
        xor     r8d, r8d
        xor     r9d, r9d
        xor     esi, esi
.L3:
        movzx   ecx, WORD PTR [rdi+rdx*2]
        mov     eax, ecx
        neg     ax
        cmovs   eax, ecx
        movzx   eax, ax
        cmp     eax, esi
        jle     .L2
        mov     r8d, edx
        movsx   r9d, cx
.L2:
        cmp     esi, eax
        cmovl   esi, eax
        add     rdx, 1
        cmp     rdx, 128
        jne     .L3
        xor     eax, eax
        test    r9d, r9d
        setg    al
        sal     eax, 7
        or      eax, r8d
        ret
```

RISC-V arrays comparison

rv32gc Clang 15.0, -Ofast

```
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        if(a[i] != b[i]){
            cmp = 0;
        }
    }
    return cmp;
}
```

RISC-V arrays comparison

rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        if(a[i] != b[i]){
            cmp = 0;
        }
    }
    return cmp;
}
```

```asm
array_compare: # @array_compare
  li a3, 1
  bgtz a2, .LBB0_3
.LBB0_1:
  mv a0, a3
  ret
.LBB0_2: # in Loop: Header=BB0_3 Depth=1
  addi a2, a2, -1
  addi a1, a1, 4
  addi a0, a0, 4
  beqz a2, .LBB0_1
.LBB0_3: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a0)
  lw a5, 0(a1)
  beq a4, a5, .LBB0_2
  li a3, 0
  j .LBB0_2
```

RISC-V arrays comparison

rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        if(a[i] != b[i]){
            cmp = 0;
        }
    }
    return cmp;
}
```

```asm
array_compare: # @array_compare
  li a3, 1
  bgtz a2, .LBB0_3
.LBB0_1:
  mv a0, a3
  ret
.LBB0_2: # in Loop: Header=BB0_3 Depth=1
  addi a2, a2, -1
  addi a1, a1, 4
  addi a0, a0, 4
  beqz a2, .LBB0_1
.LBB0_3: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a0)
  lw a5, 0(a1)
  beq a4, a5, .LBB0_2
  li a3, 0
  j .LBB0_2
```

RISC-V Comparisons

C, rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        cmp &= a[i] == b[i];
    }
    return cmp;
}
```

RISC-V Comparisons

C, rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        cmp &= a[i] == b[i];
    }
    return cmp;
}
```

```asm
array_compare: # @array_compare
  blez a2, .LBB0_4
  mv a3, a0
  li a0, 1
.LBB0_2: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a3)
  lw a5, 0(a1)
  xor a4, a4, a5
  seqz a4, a4
  and a0, a0, a4
  addi a2, a2, -1
  addi a1, a1, 4
  addi a3, a3, 4
  bnez a2, .LBB0_2
  ret
.LBB0_4:
  li a0, 1
  ret
```

RISC-V Comparisons

C, rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        cmp &= a[i] == b[i];
    }
    return cmp;
}
```

```asm
array_compare: # @array_compare
  blez a2, .LBB0_4
  mv a3, a0
  li a0, 1
.LBB0_2: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a3)
  lw a5, 0(a1)
  xor a4, a4, a5
  seqz a4, a4
  and a0, a0, a4
  addi a2, a2, -1
  addi a1, a1, 4
  addi a3, a3, 4
  bnez a2, .LBB0_2
  ret
.LBB0_4:
  li a0, 1
  ret
```

RISC-V Comparisons

C, rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        cmp *= a[i] == b[i];
    }
    return cmp;
}
```

RISC-V Comparisons

C, rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        cmp *= a[i] == b[i];
    }
    return cmp;
}
```

```
array_compare: # @array_compare
  li a3, 1
  bgtz a2, .LBB0_3
.LBB0_1:
  mv a0, a3
  ret
.LBB0_2: # in Loop: Header=BB0_3 Depth=1
  addi a2, a2, -1
  addi a1, a1, 4
  addi a0, a0, 4
  beqz a2, .LBB0_1
.LBB0_3: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a0)
  lw a5, 0(a1)
  beq a4, a5, .LBB0_2
  li a3, 0
  j .LBB0_2
```

RISC-V Comparisons

C, rv32gc Clang 15.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size) {
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        cmp *= a[i] == b[i];
    }
    return cmp;
}
```

```asm
array_compare: # @array_compare
  li a3, 1
  bgtz a2, .LBB0_3
.LBB0_1:
  mv a0, a3
  ret
.LBB0_2: # in Loop: Header=BB0_3 Depth=1
  addi a2, a2, -1
  addi a1, a1, 4
  addi a0, a0, 4
  beqz a2, .LBB0_1
.LBB0_3: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a0)
  lw a5, 0(a1)
  beq a4, a5, .LBB0_2
  li a3, 0
  j .LBB0_2
```

RISC-V Comparisons

C, rv32gc Clang 16.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        if(a[i] != b[i]){
            cmp = 0;
        }
    }
    return cmp;
}
```

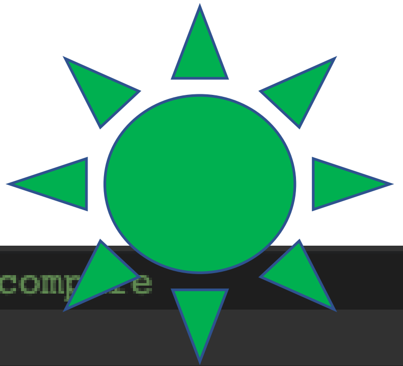
RISC-V Comparisons

C, rv32gc Clang 16.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        if(a[i] != b[i]){
            cmp = 0;
        }
    }
    return cmp;
}
```

```
array_compare: # @array_compare
  blez a2, .LBB0_4
  mv a3, a0
  li a0, 1
.LBB0_2: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a3)
  lw a5, 0(a1)
  xor a4, a4, a5
  snez a4, a4
  addi a4, a4, -1
  and a0, a0, a4
  addi a2, a2, -1
  addi a1, a1, 4
  addi a3, a3, 4
  bnez a2, .LBB0_2
  ret
.LBB0_4:
  li a0, 1
  ret
```

RISC-V Comparisons



C, rv32gc Clang 16.0, -Ofast

```c
uint32_t array_compare(uint32_t *a, uint32_t *b, int size){
    uint32_t cmp = 1;
    for (int i = 0; i < size; ++i) {
        if(a[i] != b[i]){
            cmp = 0;
        }
    }
    return cmp;
}
```

```asm
array_compare: # @array_compare
  blez a2, .LBB0_4
  mv a3, a0
  li a0, 1
.LBB0_2: # =>This Inner Loop Header: Depth=1
  lw a4, 0(a3)
  lw a5, 0(a1)
  xor a4, a4, a5
  snez a4, a4
  addi a4, a4, -1
  and a0, a0, a4
  addi a2, a2, -1
  addi a1, a1, 4
  addi a3, a3, 4
  bnez a2, .LBB0_2
  ret
.LBB0_4:
  li a0, 1
  ret
```