Next Generation Crypto

> Light-weight Cryptography.
Post Quantum Cryptography
Zero-knowledge Proofs.
Homomorphic Encryption.
IPFS

Prof Bill Buchanan OBE
https://asecuritysite.com/

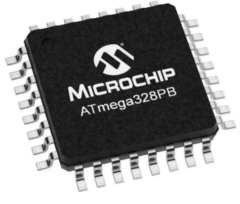
Next Generation Cry

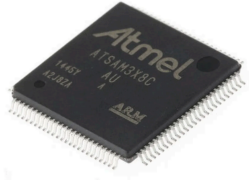
> Light-weight Cryptograph
Post Quantum Cryptograph
Zero-knowledge Proofs.
Homomorphic Encryption.
IPFS

Prof Bill Buchanan O
https://asecuritysite.com/

No	Date	Subject	Lab
2	27 Jan 2025	Ciphers and Fundamentals [Unit]	[Lab] [Demo]
3	3 Feb 2025	Symmetric Key [Unit]	[Lab] Vincent Rijmen
4	10 Feb 2025	Hashing and MAC [Unit]	[Lab] Ivan Damgård
5	17 Feb 2025	Asymmetric (Public) Key [Unit]	[Lab] Len Adleman
6	24 Feb 2025	Key Exchange [Unit]	[Lab] Whitfield Diffie
7	3 Mar 2025	Reading Week (Revision lecture)	Mini-project [Here] / Coursework
8	10 Mar 2025	Digital Signatures and Certificates [Unit]	[Lab]
9	17 Mar 2025	Test (Units 1-5) 40% of overall mark [Here]	
10	24 Mar 2025	Tunnelling [Unit]	[Lab Marty Hellman
11	31 Mar 2025	Blockchain [Unit]	[Lab] Troy Hunt
12	7 Apr 2025	Future Cryptography [Unit]	[Lab]
13	28 Apr 2025	Host/Cloud Security [Unit]	[Lab]
14	5 May 2025	Coursework Hand-in - 60% of overall mark (Sunday, 11 May 2025) [Coursework]	Daniel J Bernstein

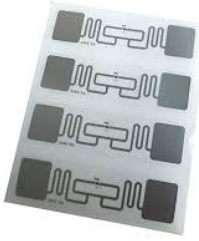
A Computer?





Light-weight crypto

- Conventional cryptography. Servers and Desktops. Tablets and smart phones.
- Light-weight cryptography. Embedded Systems. RFID and Sensor Networks.



Thus often light-weight cryptography methods balance performance (throughput) against **power drain and GE (gate equivalents).** Along with this the method must also have a low requirement for RAM (where the method requires the usage of running memory to perform its operation) and ROM (where the method is stored on the device). In order to assess the strengths of various methods we often **define the area** that the cryptography function will use on the device — and which is defined in μm^2.

View methods

Light-weight crypto

Cipher	Key bits	Block bits	Cycles per block	Throughput at 100 kHz (Kbps)	Logic process	Area (GEs)
Block ciphers						
Present	80	64	32	200.00	0.18 µm	1,570
AES	128	128	1,032	12.40	0.35 µm	3,400
Hight	128	64	34	188.20	0.25 µm	3,048
Clefia	128	128	36	355.56	0.09 µm	4,993
mCrypton	96	64	13	492.30	0.13 µm	2,681
DES	56	64	144	44.40	0.18 µm	2,309
DESXL	184	64	144	44.40	0.18 µm	2,168
Stream ciphers						
Trivium[5]	80	1	1	100.00	0.13 µm	2,599
Grain[5]	80	1	1	100.00	0.13 µm	1,294

*AES: Advanced Encryption Standard; DES: Data Encryption Standard; DESXL: lightweight DES with key whitening.

function will use on the device — and which is defined in μm^2.

View methods

Light-weight crypto

Cipher	Key size (bits)	Block size (bits)	Encryption (cycles/ block)	Throughput at 4 MHz (Kbps)	Decryption (cycles/ block)	Relative throughput (% of AES)	Code size (bytes)	SRAM size (bytes)	Relative code size (% of AES)
Hardware-oriented block ciphers									
DES	56	64	8,633	29.6	8,154	38.4	4,314	0	152.4
DESXL	184	64	8,531	30.4	7,961	39.4	3,192	0	112.8
Hight	128	64	2,964	80.3	2,964	104.2	5,672	0	200.4
Present	80	64	10,723	23.7	11,239	30.7	936	0	33.1
Software-oriented block ciphers									
AES	128	128	6,637	77.1	7,429	100.0	2,606	224	100.0
IDEA	128	64	2,700	94.8	15,393	123.0	596	0	21.1
TEA	128	64	6,271	40.8	6,299	53.0	1,140	0	40.3
SEA	96	96	9,654	39.7	9,654	51.5	2,132	0	75.3
Software-oriented stream ciphers									
Salsa20	128	512	18,400	111.3	NA	144.4	1,452	280	61.2
LEX	128	320	5,963	214.6	NA	287.3	1,598	304	67.2

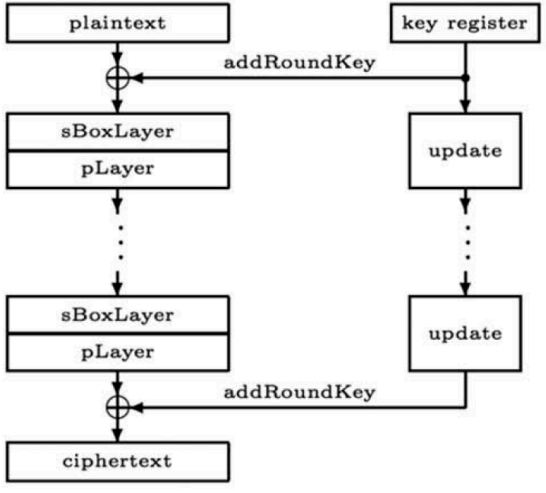
*IDEA: International Data Encryption Algorithm; TEA: Tiny Encryption Algorithm; SEA: Scalable Encryption Algorithm.

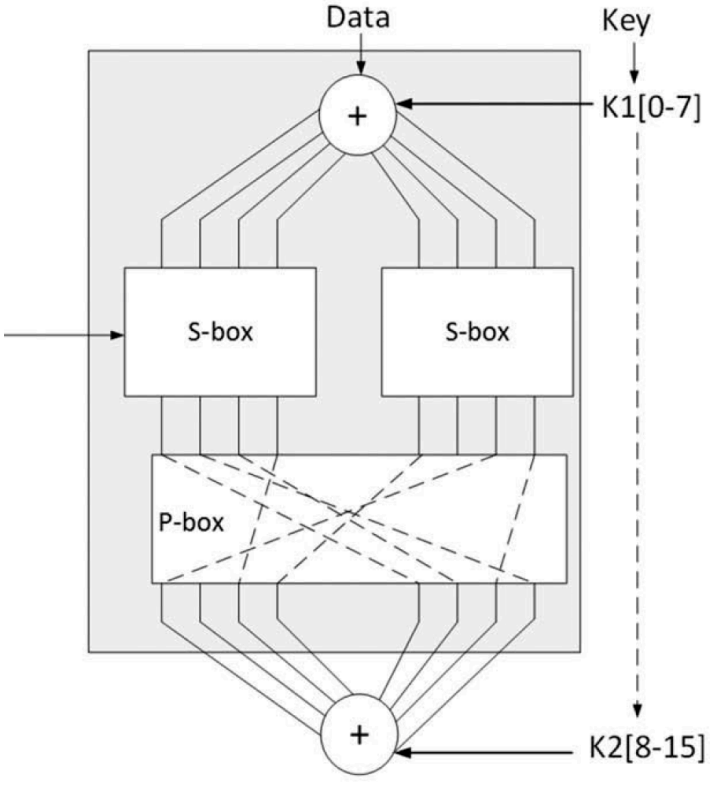
function will use on the device" and which is defined in *pm*.

View methods

Light-weight crypto (PRESENT)

```
generateRoundKeys()
for i = 1 to 31 do
    addRoundKey(STATE, K_i)
    sBoxLayer(STATE)
    pLayer(STATE)
end for
addRoundKey(STATE, K_32)
```





x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S[x]$	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

View methods

Light-weight crypto (Finalists)

	Area (μm^2 - synthesis over 22nm)	Area (kGE)
TinyJambu	716	3.6
Grain128-AEAD	861	4.3
GIFT-COFB	1,618	8.1
Romulus	1,961	9.8
ASCON	21,93	11
Xoodyak	2,387	11.9
Photon-Beetle	2,523	12.6
ISAP	3,080	15.4
Elephant	3,458	17.3
Sparkle	7,897	739.5

Name	Type	Variant	Underlying Primitive	State (Bits)	Key (Bits)	Mode	Rate/Block (Bits)	Tag (Bits)	Security (Bits)
Ascon	Sponge	Ascon-128	Ascon-p	320	128	Duplex	64	128	128
		Ascon-128a	Ascon-p	320	128	Duplex	128	128	128
Elephant	Sponge	Jumbo	Spongent	176	128	Elephant	176	64	127
		Dumbo	Spongent	160	128	Elephant	160	64	112
		Delirium	Keccak	200	128	Elephant	176	128	127
GIFT-COFB	Block	GIFT-COFB	GIFT-128	192	128	COFB	128	128	128
Grain-128AEAD	Stream	Grain-128AEAD	N/A	256	128	N/A	1	64	128
ISAP	Sponge	ISAP-A-128	Ascon-p	320	128	ISAP	64	128	128
		ISAP-K-128	Keccak	400	128	ISAP	144	128	128
		ISAP-K-128A	Keccak	400	128	ISAP	144	128	128
		ISAP-A-128A	Ascon-p	320	128	ISAP	64	128	128
PHOTON-Beetle	Sponge	PHOTON-Beetle-AEAD[128]	PHOTON256	256	128	Beetle	128	256	121
		PHOTON-Beetle-AEAD	PHOTON256	256	128	Beetle	32	256	128
Romulus	Block	Romulus-M	Skinny-128-384	384	128	COFB	128	128	128
		Romulus-N	Skinny-128-384	384	128	COFB	128	128	128
		Romulus-T	Skinny-128-384	384	128	COFB	128	128	128
SPARKLE	Sponge	SCHWAEMM256-128	SPARKLE	384	128	SPARKLE	256	128	120
		SCHWAEMM128-128	SPARKLE	256	128	SPARKLE	128	128	120
		SCHWAEMM192-192	SPARKLE	384	192	SPARKLE	192	192	184
		SCHWAEMM256-256	SPARKLE	512	256	SPARKLE	256	256	248
TinyJambu	Sponge	TinyJambu	TinyJambu	128	128	TinyJambu	32	64	120
Xoodyak	Sponge	Xoodyak	Xoodoo	384	128	Cyclist	352	128	128

Elsadek, S. Aftabjahani, D. Gardner, E. MacLean, J. R. Wallrabenstein, and E. Y. Tawfik, "Hardware and energy efficiency evaluation of nist lightweight cryptography standardization finalists," in 2022 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2022, pp. 133–137.
H. Madushan, I. Salam, and J. Alawatugoda, "A review of the nist lightweight cryptography finalists and their fault analyses," Electronics, vol. 11, no. 24, p. 4199, 2022.

View methods

Light-weight crypto (Finalists)

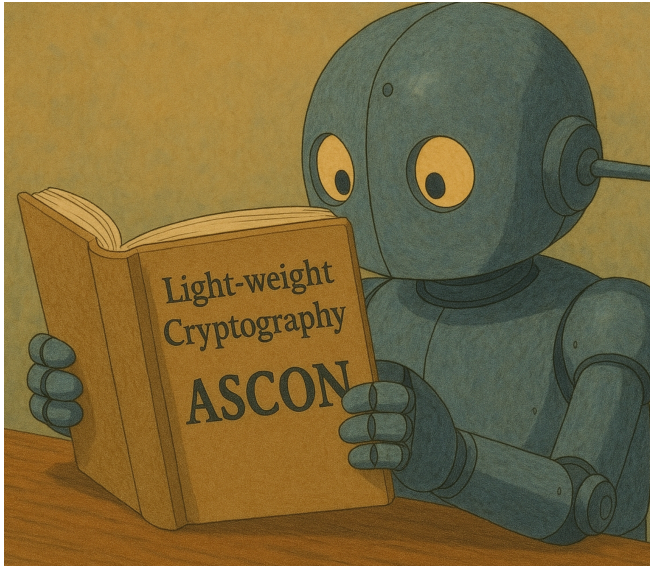
Algorithm	Impl.	Primary	Flag	Size	h(8)	h(16)	h(32)	h(64)	h(128)	Benchmark
reference	$nacl_ref$	yes	O3	18774	768	768	772	1364	1968	1
sparkle	rhys	yes	O1	7912	1036	1036	1468	2272	3884	2
Xoodyak	XKCP-AVR8	yes	O3	2604	1284	1288	1924	3192	5732	2.9
gimli	rhys	yes	O3	19554	1284	1920	2544	3804	6312	3.2
ascon	rhys	yes	O3	2178	2972	3552	4736	7088	11784	6
drygascon	rhys	no	O3	15500	4604	4600	6540	10360	17912	9.1
photon-beetle	$avr8_speed$	yes	O3	2948	2372	2364	6940	16084	34172	17.4
skinny	rhys	yes	O2	9784	7048	10556	13976	20952	34896	17.7

Algorithm	Impl.	Primary	Flag	Size	Enc(0:8)	Dec(0:8)	Enc(128:129)	Dec(128:128)	Bench.(128)	Bench.(8)
sparkle	rhys	yes	O3	12290	1276	1316	4648	5072	4.7	3.3
Xoodyak	XKCP-AVR8	yes	O3	4560	2596	2608	7184	7128	3.3	1.6
knot	$avr8_speed$	no	Os	1664	2124	2140	8144	8160	2.9	2
ascon	rhys	no	O3	5180	1240	1284	8056	8488	2.8	3.3
GIFT-COFB	rhys	yes	O1	23312	1852	1892	8220	8776	2.7	2.2
saeaes	ref	no	O3	17062	1208	1212	8992	9004	2.6	3.4
hyena	rhys	yes	O3	293860	1912	1964	8960	9396	2.5	2.2
elephant	rhys	no	O3	13106	1924	1948	9260	9796	2.4	2.2
estate	ref	yes	O3	9434	1424	1448	10276	10292	2.3	2.9
romulus	rhys	no	O3	19346	1632	1676	10152	10568	2.2	2.5
spook	rhys	no	O3	12942	2984	2968	10272	10708	2.2	1.4
tinyjambu	rhys	yes	O3	9174	1232	1288	10364	10888	2.2	3.4
subterranean	rhys	yes	Os	6042	3372	3460	10288	10944	2.2	1.2
orange	rhys	yes	O3	12140	2500	2536	11200	11620	2	1.7
gimli	rhys	yes	O3	21272	1920	1956	11944	12360	1.9	2.2
skinny	rhys	no	O1	12452	1604	1644	12960	14372	1.7	2.6
photon-beetle	$avr8_speed$	yes	Os	3536	2444	2472	20076	20092	1.2	1.7
reference	rhys	yes	O2	7874	4152	4156	23812	23764	1	1
grain128aead	rhys	yes	O2	9532	3992	3980	30396	30124	0.8	1
isap	rhys	no	O2	3824	20212	20256	42936	43372	0.5	0.2

E. Bellini and Y. J. Huang, “Randomness testing of the nist light weight cipher finalist candidates,” in NIST Lightweight Cryptography Workshop, May, 2022.

View methods

Methods



- ASCON with Bouncy Castle. https://asecuritysite.com/ascon/bc_ascon
- ASCON AEAD: https://asecuritysite.com/light/ascon01

Next Generation Crypto

Light-weight Cryptography.
> Post Quantum Cryptography
Zero-knowledge Proofs.
Homomorphic Encryption.
IPFS

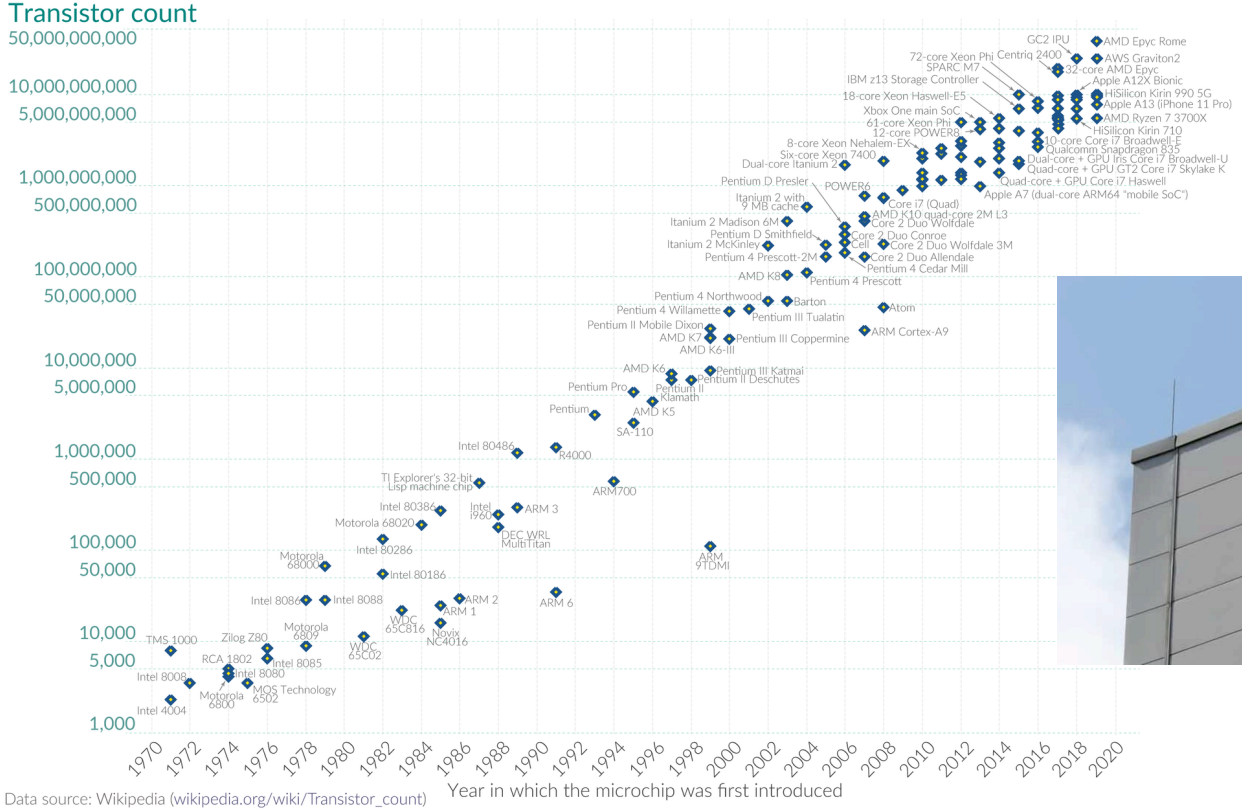
Prof Bill Buchanan OBE
https://asecuritysite.com/pqc

Moore's Law

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

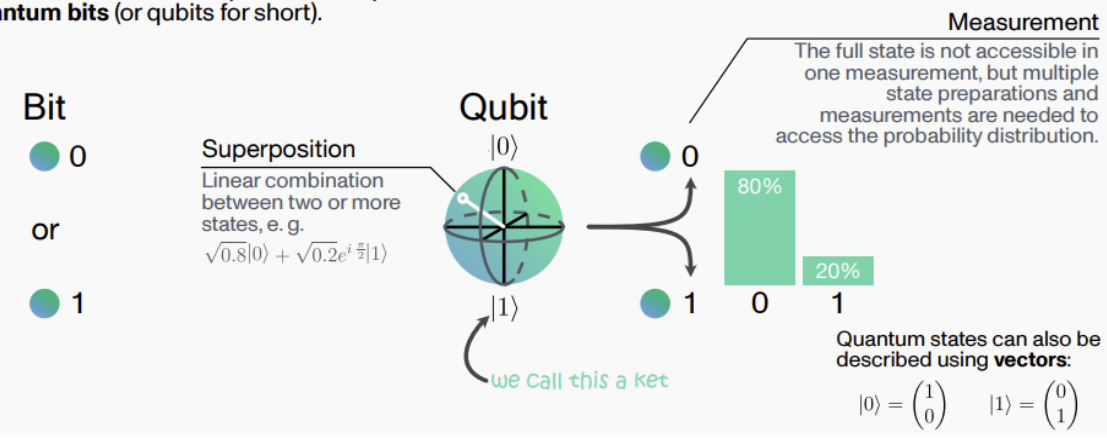


Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

Qubits.

Bits and Qubits

Instead of classical bits, quantum computers use **quantum bits** (or qubits for short).

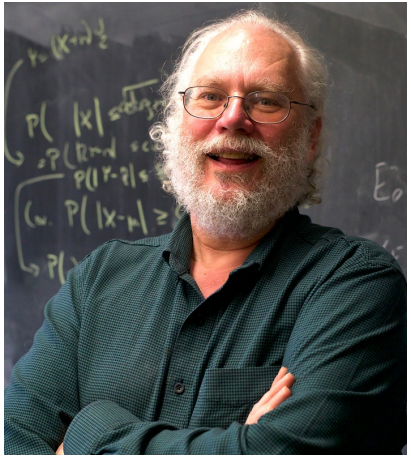


One-Qubit Gates

Gate		Matrix
X	**Pauli-X** is a 180° rotation around the x-axis; also known as the quantum NOT gate	$\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$
Y	**Pauli-Y** is a 180° rotation around the y-axis	$\begin{pmatrix}0 & -i\\i & 0\end{pmatrix}$
Z	**Pauli-Z** is a 180° rotation around the z-axis	$\begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}$
H	**Hadamard** maps $\|0\rangle$ to $\|+\rangle$ and $\|1\rangle$ to $\|-\rangle$; used to create an equal superposition	$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$
S	**S** is a 90° rotation around the z-axis; S S = Z ; The inverse $S^\dagger$ rotates in the opposite direction	$\begin{pmatrix}1 & 0\\0 & i\end{pmatrix}$

Superposition

Shor's algorithm



See Appendix 1 for details + Colab notebook for Shor.

Factoring through order-finding

The following method succeeds in finding a factor of N with probability at least $1/2$, provided N is odd and not a prime power.

Factor-finding method

1. Choose $a \in \{2, \ldots, N-1\}$ at random.
2. Compute $d = \gcd(a, N)$. If $d \geq 2$ then output d and stop.
3. *Compute the order* r of a modulo N.
4. If r is even, then compute $d = \gcd(a^{r/2} - 1, N)$. If $d \geq 2$, output d and stop.
5. If this step is reached, the method has failed.

Main idea

1. By the definition of the order, we know that $a^r \equiv 1 \pmod{N}$.

$$a^r \equiv 1 \pmod{N} \quad \Leftrightarrow \quad N \text{ divides } a^r - 1$$

2. If r is even, then

$$a^r - 1 = (a^{r/2} + 1)(a^{r/2} - 1)$$

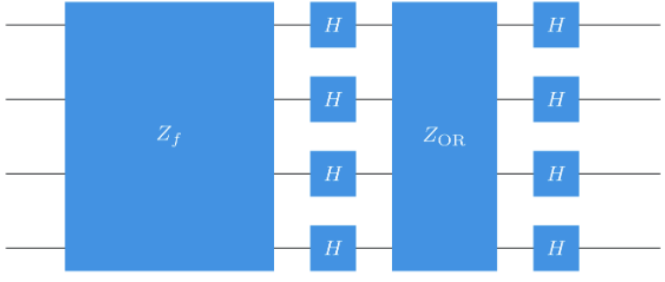
Each prime dividing N must therefore divide either $(a^{r/2} + 1)$ or $(a^{r/2} - 1)$.
For a random a, at least one of the prime factors of N is likely to divide $(a^{r/2} - 1)$.

Grover's algorithm.

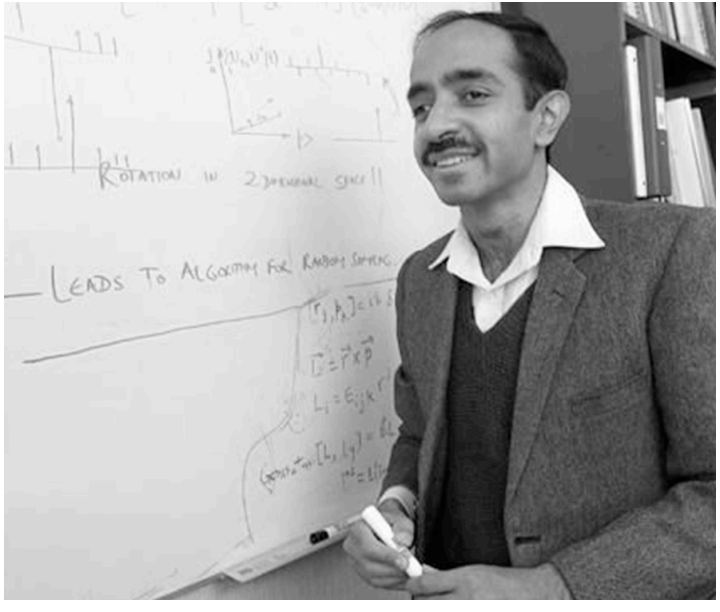
Grover's algorithm

1. Initialize an n qubit register Q to the all-zero state $|0^n\rangle$ and then apply a Hadamard operation to each qubit of Q.
2. Apply t times the unitary operation $G = H^{\otimes n} Z_{\mathrm{OR}} H^{\otimes n} Z_f$ to the register Q
3. Measure the qubits of Q with respect to standard basis measurements and output the resulting string.

The operation $G = H^{\otimes n} Z_{\mathrm{OR}} H^{\otimes n} Z_f$ iterated in step 2 will be called the *Grover operation* throughout the remainder of this lesson. Here is a quantum circuit representation of the Grover operation:



Here the Z_f operation is depicted as being larger than Z_{OR} as a way to suggest that it is likely to be the more costly operation (but this is only meant a visual clue and not something with a formal meaning). In particular, when we're working within the query model Z_f requires one query while Z_{OR} requires no queries — and if instead we have a Boolean circuit for the function f and convert it to a quantum circuit for Z_f, we can reasonably expect that the resulting quantum circuit will be larger and more complicated than one for Z_{OR}.



See Appendix 2 for details + Colab notebook for Grover.

Why worry?

President Biden Signs Memo to Combat Quantum Computing Threat

FORT MEADE, Md. — The White House announced today that President Joe Biden has signed a National Security Memorandum (NSM) aimed at maintaining U.S. leadership in quantum information sciences and to mitigate the risks of quantum computing to the Nation's security.

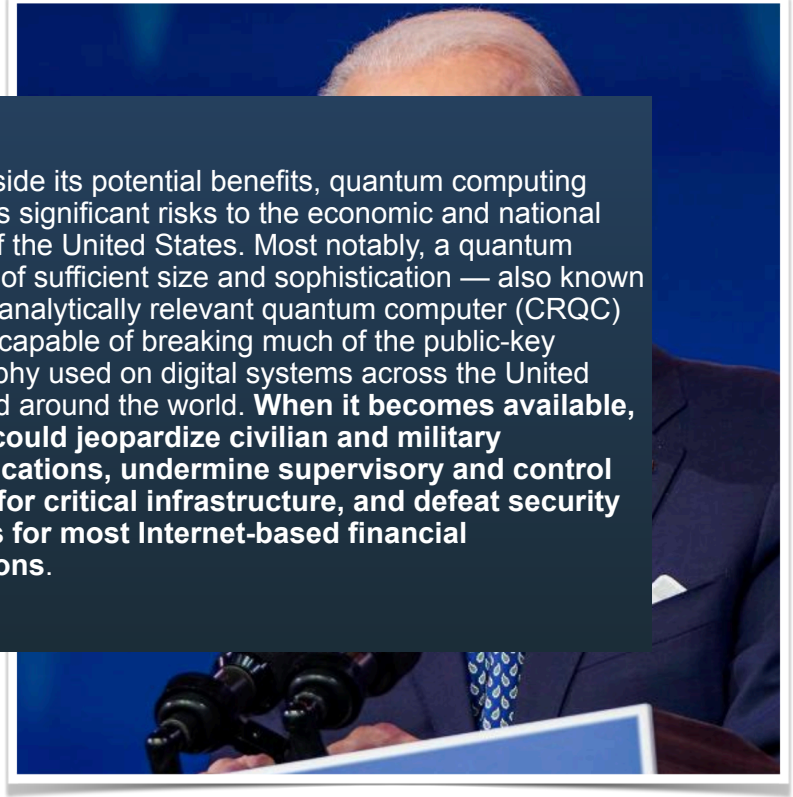
"Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems" - also known as NSM-10 - directs U.S. Government agencies to migrate vulnerable cryptographic systems to quantum-resistant cryptography as part of multi-year effort. As the National Manager for National Security Systems, the Director of NSA will oversee this process across the 50-plus government departments and agencies using National Security Systems (NSS) - systems that contain classified information or are otherwise critical to military or intelligence operations.

Why worry?

President Biden Signs Memo to Combat Quantum Computing Threat

FORT MEADE, Md. — The White House announced today that President Joe Biden has signed a National Security Memorandum (NSM) aimed at maintaining U.S. leadership in quantum information sciences and to mitigate the risks of quantum computing to the Nation's security.

"Promoting United States Leadership in Quantum Computing While Mitigating Risks to Vulnerable Cryptographic Systems" - also known as NSM-10 - directs U.S. Government agencies to migrate vulnerable cryptographic systems to quantum-resistant cryptography as part of multi-year effort. As the National Manager for National Security Systems, the Director of NSA will oversee this process across the 50-plus government departments and agencies using National Security Systems (NSS) - systems that contain classified information or are otherwise critical to military or intelligence operations.



Yet alongside its potential benefits, quantum computing also poses significant risks to the economic and national security of the United States. Most notably, a quantum computer of sufficient size and sophistication — also known as a cryptanalytically relevant quantum computer (CRQC) — will be capable of breaking much of the public-key cryptography used on digital systems across the United States and around the world. **When it becomes available, a CRQC could jeopardize civilian and military communications, undermine supervisory and control systems for critical infrastructure, and defeat security protocols for most Internet-based financial transactions**.

Why worry?

President Biden Signs Memo to Combat Quantum Computing Threat

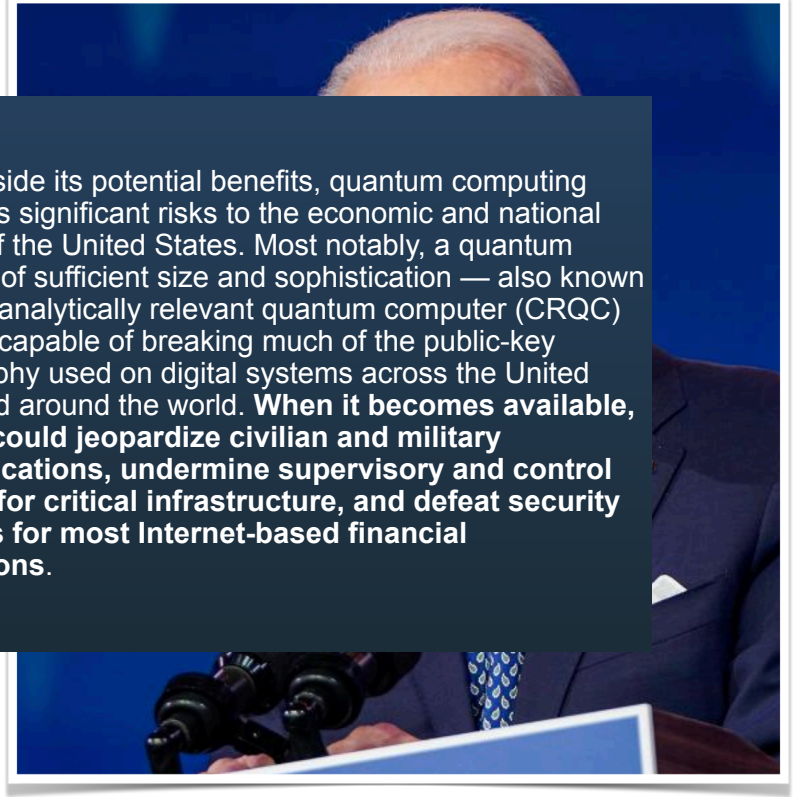
Within 90 days of the release of the first set of NIST standards for quantum-resistant cryptography referenced in subsection 3(a) of this memorandum, and on an annual basis thereafter, as needed, the Secretary of Commerce, through the **Director of NIST, shall release a proposed timeline for the deprecation of quantum-vulnerable cryptography in standards**, with the goal of moving the maximum number of systems off quantum-vulnerable cryptography within a decade of the publication of the initial set of standards. The Director of NIST shall work with the appropriate technical standards bodies to encourage interoperability of commercial cryptographic approaches.

t President
M) aimed a
s and to
urity.

While
known as
rable
s part of
rity System
lus
rity Systems
herwise

critical to military or intelligence operations.

Yet alongside its potential benefits, quantum computing also poses significant risks to the economic and national security of the United States. Most notably, a quantum computer of sufficient size and sophistication — also known as a cryptanalytically relevant quantum computer (CRQC) — will be capable of breaking much of the public-key cryptography used on digital systems across the United States and around the world. **When it becomes available, a CRQC could jeopardize civilian and military communications, undermine supervisory and control systems for critical infrastructure, and defeat security protocols for most Internet-based financial transactions**.



Why worry?

Pres
Cor

Within 90 days
standards for q
subsection 3(a)
thereafter, as n
the **Director of**
the deprecatio
standards, wit
systems off qua
decade of the p
Director of NIS
standards bodi
cryptographic a

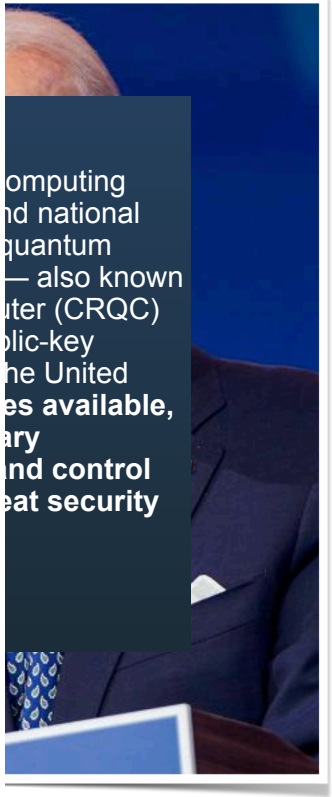
critical to mili

QUANTUM-READINESS: MIGRATION TO POST-QUANTUM CRYPTOGRAPHY

NIST NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY U.S. DEPARTMENT OF COMMERCE

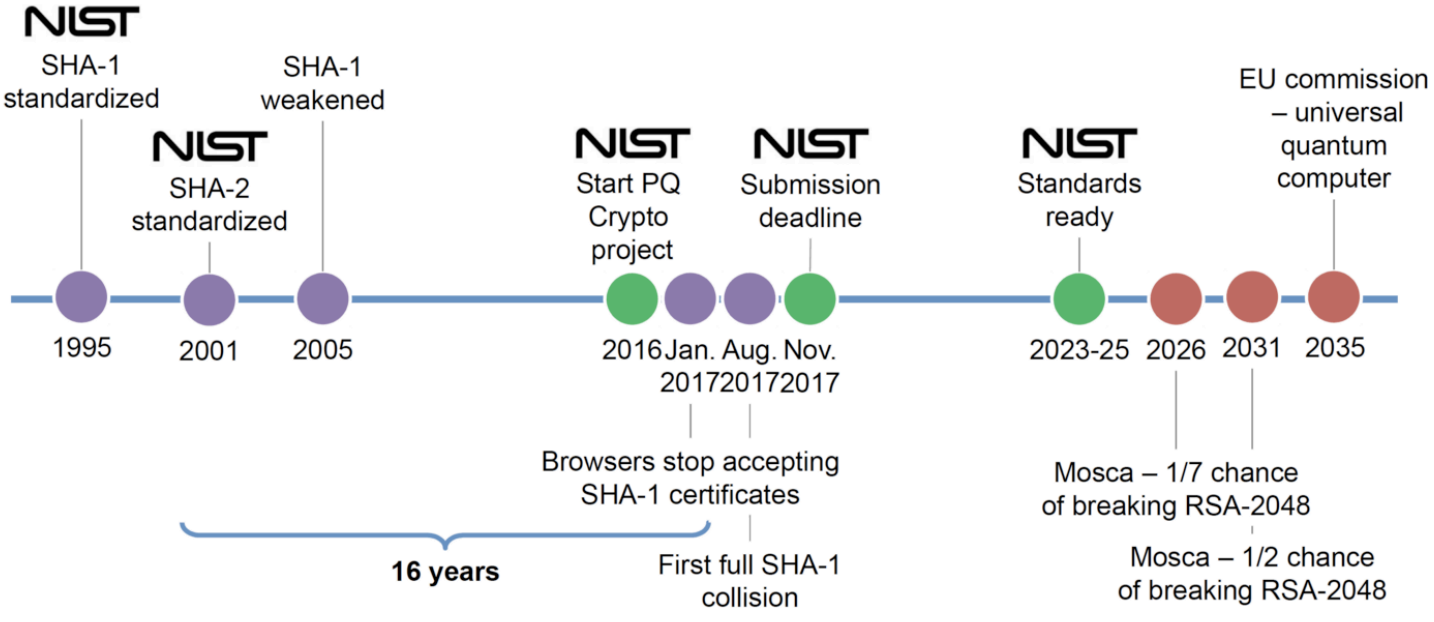
BACKGROUND

The Cybersecurity and Infrastructure Security Agency (CISA), the National Security Agency (NSA), and the National Institute of Standards and Technology (NIST) created this factsheet to inform organizations — especially those that support Critical Infrastructure — about the impacts of quantum capabilities, and to encourage the early planning for migration to post-quantum cryptographic standards by developing a Quantum-Readiness Roadmap. NIST is working to publish the first set of post-quantum cryptographic (PQC) standards, to be released in 2024, to protect against future, potentially adversarial, cryptanalytically-relevant quantum computer (CRQC) capabilities. A CRQC would have the potential to break public-key systems (sometimes referred to as asymmetric cryptography) that are used to protect information systems today.



omputing
nd national
quantum
— also known
ter (CRQC)
lic-key
he United
es available,
ary
nd control
eat security

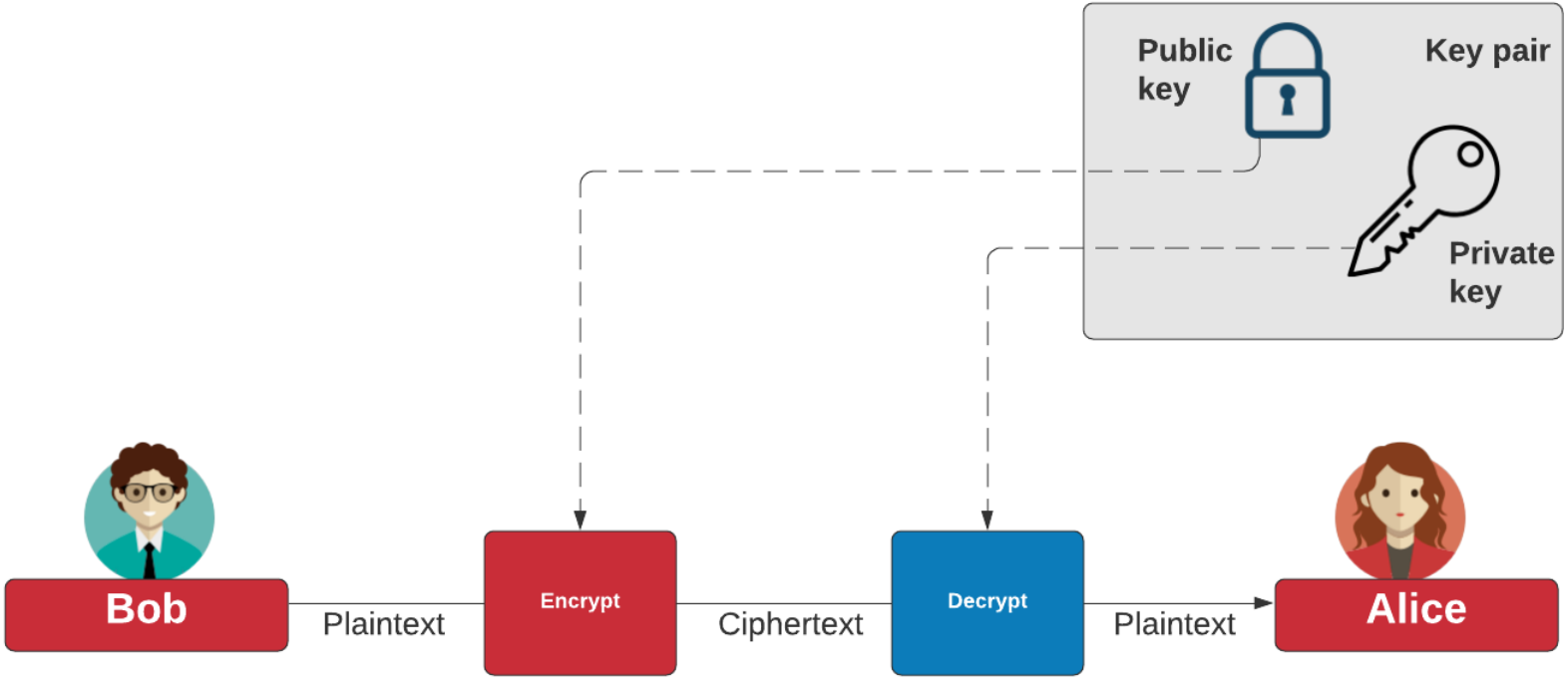
KEM and Digital Signatures/Encryption



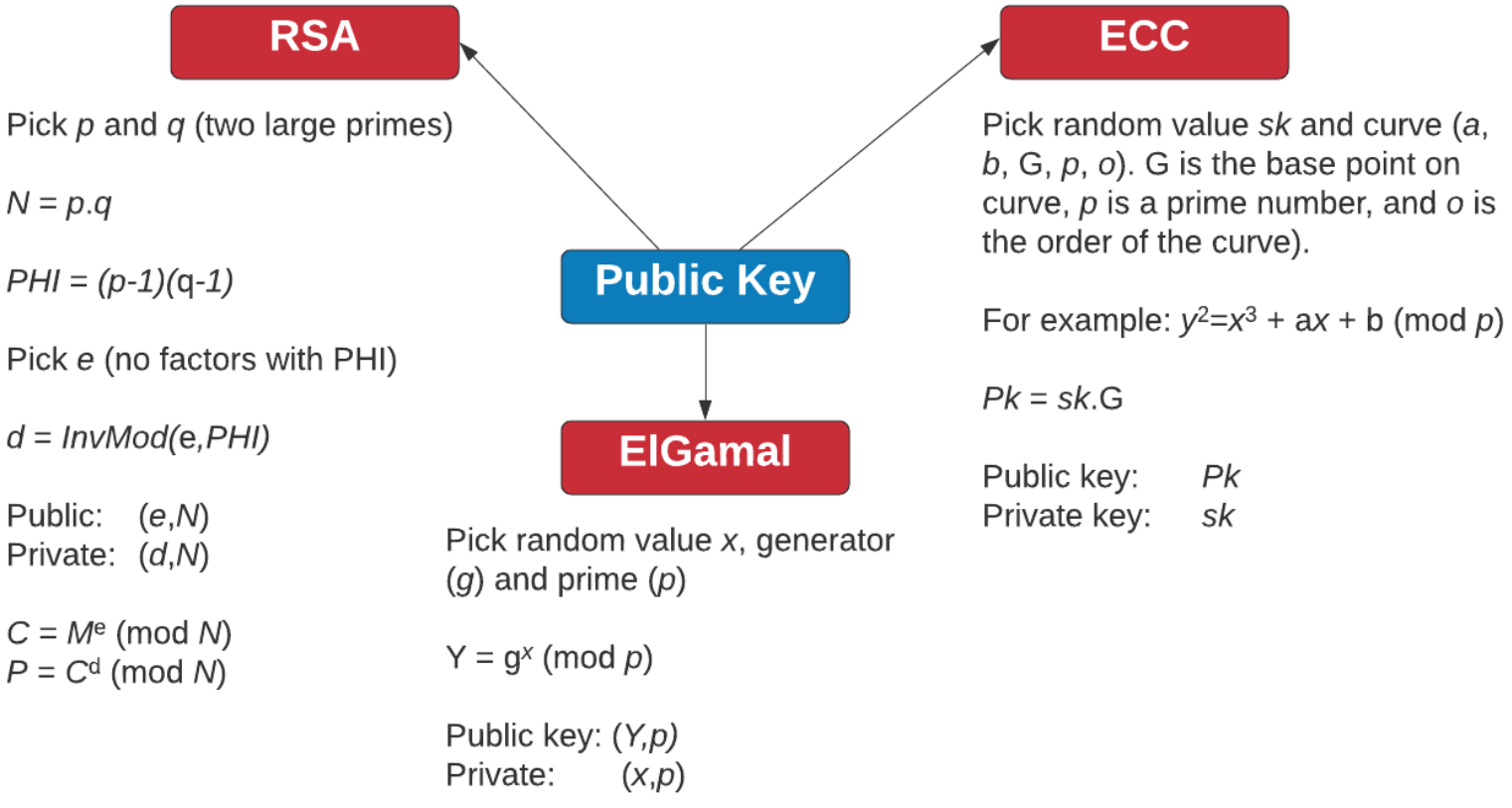
asecuritysite.com/pqc/

NIST

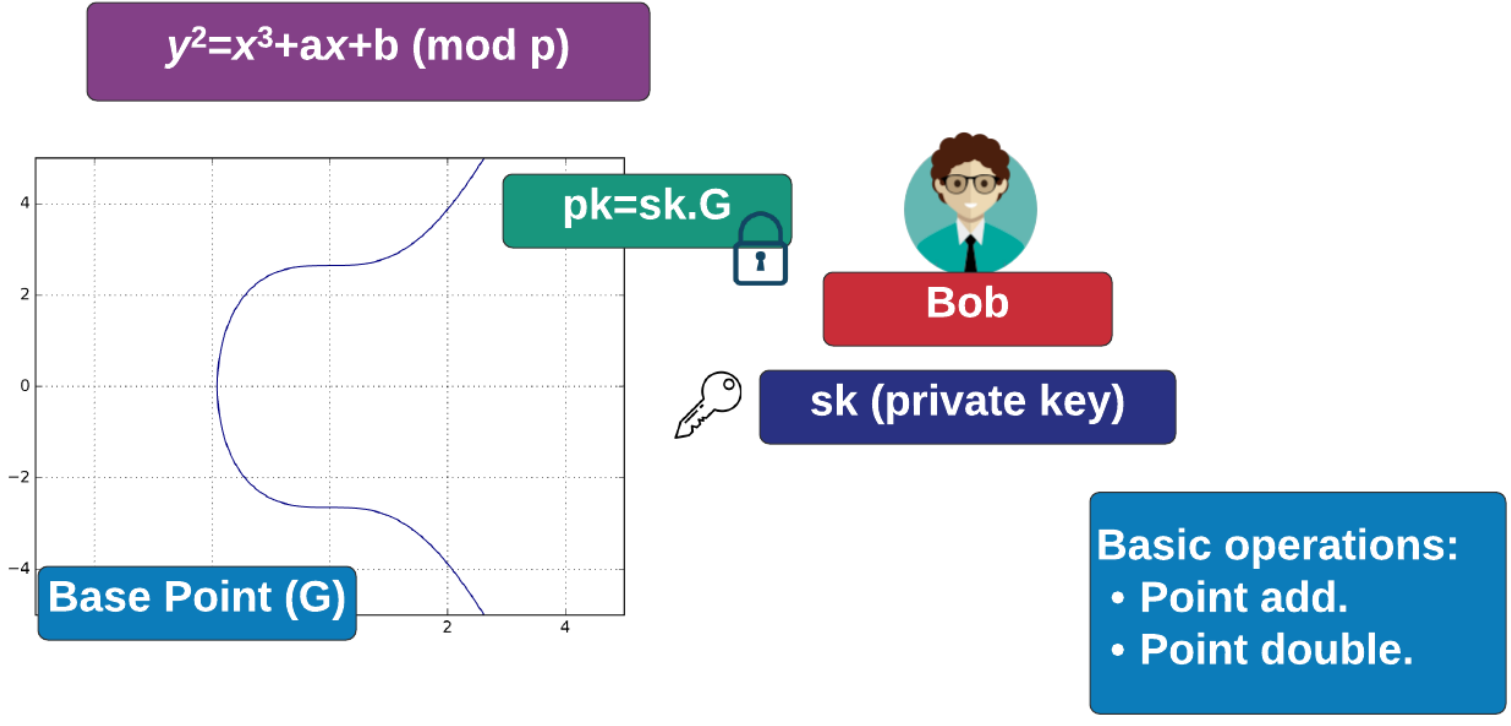
Public Key Encryption/Key Exchange



Existing Public Key Methods

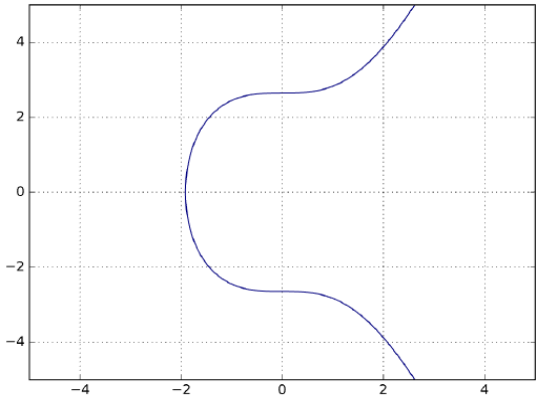


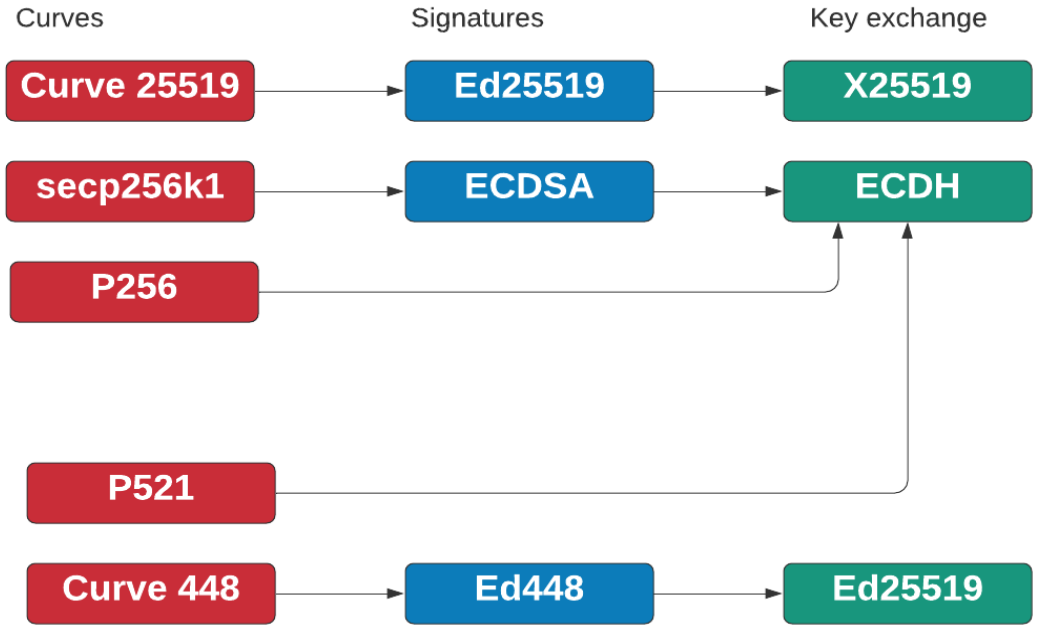
Elliptic Curve Outline



https://asecuritysite.com/ecc/ecc_types

Existing Signature and Key Exchange Methods





https://asecuritysite.com/ecc/ecc_types

KEM and Digital Signatures/Encryption

For Public-Key Encryption and KEMs (Key Exchange) we have:

- **Classic McEliece**. This has been around for around 40 years, and has been shown to be fairly resistant to attack. It produces a fairly long encryption key, but produces a fairly small amount of ciphertext.
- **CRYSTALS-KYBER (Lattice)**. Uses LWE (Learning with Errors) with lattice methods. A new lattice attack was discovered within the period of the assessment, but it is hoped that an updated version of KYBER can be produced for the final assessment. NIST have some worried about its side-channel robustness, and is a strong contender for KEM.
- **NTRU (Lattice)**. This is a traditional structured lattice based approach, and has been around for longer than the other lattice methods — showing that it is perhaps more robust against attack and against IP claims.
- **SABER (Lattice)**. This is based on modular learning with rounding, and uses lattice methods. SABER has excellent performance, and is possibly near production ready. NIST's only recommendation is that updates should perhaps consider side-channel attacks.

Alternative: BIKE (Code-based); FrodoKEM (LWE); HQC (Code-based); NTRU Prime (Lattice/NTRU); and SIKE (Isogenies).

Digital signatures:

- **CRYSTALS-DILITHIUM (Lattice)**. At present, CRYSTALS (Cryptographic Suite for Algebraic Lattices) supports two quantum robust mechanisms: Kyber for key-encapsulation mechanism (KEM) and key exchange; and Dilithium for a digital signature algorithm. CRYSTALS Dilithium uses lattice-based Fiat-Shamir schemes, and produces one of the smallest signatures of all the post-quantum methods, and with relatively small public and private key sizes.
- **FALCON (Lattice)**. Falcon is one of the finalists for the NIST standard for PQC (Post Quantum Cryptography), along with NTRU (Nth degree-truncated polynomial ring units) and CRYSTALS-DILITHIUM. It is derived from NTRU and is a lattice-based method for quantum robust digital signing. Falcon is based on the Gentry, Peikert and Vaikuntanathan method for generating lattice-based signature schemes, along with a trapdoor sampler - Fast Fourier sampling.
- **SPHINCS+ (Hash-based signature).**

Alternative: GeMSS (Multivariate); Picnic (Hash-based ZKP); and SPHINCS+ (Hash-based stateless).

asecuritysite.com/pqc/

NIST

Four Generations of HE

PQC Fourth Round Candidate Key-Establishment Mechanisms (KEMs)

The following candidate KEM algorithms will advance to the fourth round:

Public-Key Encryption/KEMs
BIKE
Classic McEliece
HQC
SIKE

In the first round, we have:

- **Code-based Signatures:** CROSS (Codes and Restricted Objects Signature Scheme); Enhanced pqsigRM; FuLeeca; LESS (Linear Equivalence Signature Scheme) and MEDS (Matrix Equivalence Digital Signature Wave).
- **Isogenies:** SQIsign.
- **Lattice based:** EagleSign; EHTv3 and EHTv4; HAETAE; HAWK; HuFu (Hash-and-Sign Signatures From Powerful Gadgets); Raccoon; and SQUIRRELS (Square Unstructured Integer Euclidean Lattice Signature).
- **MPC in the head:** MIRA; MiRitH (MinRank in the Head); MQOM (MQ on my Mind); PERK; RYDE; and SDitH (Syndrome Decoding in the Head).
- **Multivariate Signatures (Oil and Vinegar):** 3WISE; Biscuit; DME-Sign; HPPC (Hidden Product of Polynomial Composition); MAYO; PROV (PRovable unbalanced Oil and Vinegar); QR-UOV; SNOVA; TUOV (Triangular Unbalanced Oil and Vinegar); UOV (Unbalanced Oil and Vinegar); and VOX.
- **Symmetric-based Signatures:** AIMer; Ascon-Sign; FAEST; and SPHINCS-alpha.

Doing a quick count, we have:

Multivariate: 11; Lattice: 7; Code-based: 5; MPC-in-the-head: 5; Symmetric-based: 4; and Isogenies: 1.

KEM/Public Key Encryption

			Key Gen	Factor	Encap	Factor	Decap	Factor	Score (KG)	Score (Enc)	Score (Dec)	Overall
1	Kyber512-90s	Lattice (LWE)	30377	1	32386	1	19056	1	10	10	10	30
2	Kyber512	Lattice (LWE)	39455	1.3	44574	1.4	29462	1.5	10	10	10	30
3	Kyber768-90s	Lattice (LWE)	41042	1.4	44543	1.4	28685	1.5	10	10	10	30
4	Kyber1024-90s	Lattice (LWE)	53080	1.7	60272	1.9	41897	2.2	10	10	8	28
5	LightSaber-KEM	Lattice (LWE+R)	58722	1.9	59047	1.8	52131	2.7	10	10	8	28
6	Kyber768	Lattice (LWE)	59143	1.9	64410	2	45868	2.4	10	8	8	26
7	Kyber1024	Lattice (LWE)	75026	2.5	85229	2.6	65614	3.4	8	8	8	24
8	ntrulpr653	Lattice	82693	2.7	85179	2.6	92217	4.8	8	8	8	24
9	ntrulpr761	Lattice	86244	2.8	86863	2.7	94738	5	8	8	8	24
10	Saber-KEM	Lattice (LWE+R)	88626	2.9	92828	2.9	85405	4.5	8	8	8	24
11	ntrulpr857	Lattice	102071	3.4	108231	3.3	124951	6.6	8	8	8	24
12	FireSaber-KEM	Lattice (LWE+R)	126301	4.2	133791	4.1	128222	6.7	8	8	8	24
13	ntrulpr1277	Lattice	130320	4.3	135241	4.2	154612	8.1	8	8	8	24
14	NTRU-HPS-2048-509	Lattice	177211	5.8	47477	1.5	36974	1.9	8	10	10	28
15	HQC-128	Code-based	186524	6.1	305875	9.4	521661	27.4	8	8	5	21
16	NTRU-HRSS-701	Lattice	287042	9.4	44868	1.4	59252	3.1	8	10	8	26
17	NTRU-HPS-2048-677	Lattice	299514	9.9	65027	2	56855	3	8	8	8	24
18	HQC-192	Code-based	401412	13.2	690709	21.3	1096878	57.6	5	5	5	15
19	NTRU-HPS-4096-821	Lattice	406679	13.4	72840	2.2	71483	3.8	5	8	8	21
20	HQC-256	Code-based	697629	23	1213311	37.5	1966753	103.2	5	5	3	13
21	BIKE-L1	Code-based	734851	24.2	118519	3.7	5131009	269.3	5	8	3	16
22	sntrup653	Lattice	915914	30.2	75309	2.3	66592	3.5	5	8	8	21
23	sntrup761	Lattice	1051296	34.6	82427	2.5	70008	3.7	5	8	8	21
24	FrodoKEM-640-AES	Lattice (LWE)	1148982	37.8	1567672	48.4	1488040	78.1	5	5	5	15
25	sntrup857	Lattice	1237221	40.7	95152	2.9	94318	4.9	5	8	8	21
26	BIKE-L3	Code-based	2091161	68.8	299495	9.2	13853177	727	5	8	3	16
27	FrodoKEM-976-AES	Lattice (LWE)	2606237	85.8	4092307	126.4	3143510	165	5	3	3	11
28	sntrup1277	Lattice	2853083	93.9	122815	3.8	113776	6	5	8	8	21
29	FrodoKEM-640-SHAKE	Lattice (LWE)	3148720	103.7	4267582	131.8	4093162	214.8	3	3	3	9
30	FrodoKEM-1344-AES	Lattice (LWE)	4664746	153.6	5829221	180	5379988	282.3	3	3	3	9

1	Kyber512-90s	Lattice (LWE)	30
2	Kyber512	Lattice (LWE)	30
3	Kyber768-90s	Lattice (LWE)	30
4	Kyber1024-90s	Lattice (LWE)	28
5	LightSaber-KEM	Lattice (LWE+R)	28
6	NTRU-HPS-2048-509	Lattice	28
7	Kyber768	Lattice (LWE)	26
8	NTRU-HRSS-701	Lattice	26
9	Kyber1024	Lattice (LWE)	24
10	ntrulpr653	Lattice	24
11	ntrulpr761	Lattice	24
12	Saber-KEM	Lattice (LWE+R)	24
13	ntrulpr857	Lattice	24
14	FireSaber-KEM	Lattice (LWE+R)	24
15	ntrulpr1277	Lattice	24
16	NTRU-HPS-2048-677	Lattice	24
17	HQC-128	Code-based	21
18	NTRU-HPS-4096-821	Lattice	21
19	sntrup653	Lattice	21
20	sntrup761	Lattice	21
21	sntrup857	Lattice	21
22	sntrup1277	Lattice	21
23	Classic-McEliece-348864f	Code-based	18
24	BIKE-L1	Code-based	16
25	BIKE-L3	Code-based	16
26	Classic-McEliece-348864	Code-based	16
27	HQC-192	Code-based	15
28	FrodoKEM-640-AES	Lattice (LWE)	15
29	HQC-256	Code-based	13
30	NTRU-HPS-4096-1229	Lattice	13
31	NTRU-HRSS-1373	Lattice	13
32	Classic-McEliece-6688128	Code-based	13

31	SIDH-p434	Isogeny	5654091	186.1	11647942	359.7	4520865	237.2	3	3	3	9
32	SIKE-p434	Isogeny	6197253	204	10097567	311.8	10904398	572.2	3	3	3	9
33	FrodoKEM-976-SHAKE	Lattice (LWE)	7021826	231.2	9243416	285.4	9109536	478	3	3	3	9
34	SIDH-p503	Isogeny	7956605	261.9	15930604	491.9	6316223	331.5	3	3	3	9
35	SIKE-p503	Isogeny	8671491	285.5	14285437	441.1	15090345	791.9	3	3	3	9
36	SIKE-p434-compressed	Isogeny	11151196	367.1	16091778	496.9	11552400	606.2	3	3	3	9
37	SIDH-p434-compressed	Isogeny	11209821	369	16467316	508.5	5101392	267.7	3	3	3	9
38	NTRU-HPS-4096-1229	Lattice	12406657	408.4	460803	14.2	741261	38.9	3	5	5	13
39	FrodoKEM-1344-SHAKE	Lattice (LWE)	12601112	414.8	16107704	497.4	15400715	808.2	3	3	3	9
40	SIKE-p503-compressed	Isogeny	15026278	494.7	22251822	687.1	15891889	834	3	3	3	9
41	SIDH-p503-compressed	Isogeny	15242166	501.8	22995082	710	7088573	372	3	3	3	9
42	NTRU-HRSS-1373	Lattice	15922068	524.1	340073	10.5	895579	47	3	5	5	13
43	SIKE-p610	Isogeny	16516384	543.7	30973808	956.4	30794480	1616	3	3	0	6
44	SIDH-p610	Isogeny	17128219	563.9	32146290	992.6	13851706	726.9	3	3	3	9
45	SIDH-p751	Isogeny	23617554	777.5	48813951	1507.3	19233242	1009.3	3	0	0	3
46	SIKE-p751	Isogeny	26485988	871.9	42886151	1324.2	47206045	2477.2	3	0	0	3
47	SIDH-p610-compressed	Isogeny	30775700	1013.1	44161525	1363.6	15152561	795.2	0	0	3	3
48	SIKE-p610-compressed	Isogeny	33401019	1099.5	42572480	1314.5	32317016	1695.9	0	0	0	0
49	SIKE-p751-compressed	Isogeny	44187613	1454.6	68414496	2112.5	49102481	2576.7	0	0	0	0
50	SIDH-p751-compressed	Isogeny	44403930	1461.8	67441010	2082.4	21077729	1106.1	0	0	0	0
51	Classic-McEliece-348864f	Code-based	349054107	11490.7	61805	1.9	184433	9.7	0	10	8	18
52	Classic-McEliece-348864	Code-based	472549759	15556.2	81172	2.5	188381	9.9	0	8	8	16
53	Classic-McEliece-6688128	Code-based	899451646	29609.6	228951	7.1	529520	27.8	0	8	5	13
54	Classic-McEliece-460896f	Code-based	1109936192	36538.7	140594	4.3	430599	22.6	0	8	5	13
55	Classic-McEliece-6960119f	Code-based	1323860114	43581	187544	5.8	473180	24.8	0	8	5	13
56	Classic-McEliece-6688128f	Code-based	1400749128	46112.2	206238	6.4	511853	26.9	0	8	5	13
57	Classic-McEliece-8192128f	Code-based	1440024054	47405.1	202961	6.3	520936	27.3	0	8	5	13
58	Classic-McEliece-460896	Code-based	1496892143	49277.2	146998	4.5	426035	22.4	0	8	5	13
59	Classic-McEliece-6960119	Code-based	1587651773	52264.9	187845	5.8	464536	24.4	0	8	5	13
60	Classic-McEliece-8192128	Code-based	2654019753	87369.4	203049	6.3	526242	27.6	0	8	5	13

https://asecuritysite.com/pqc/

KEM/Public Key Encryption

		Level	Public	Factor	Cipher	Factor	Secret	Factor	Shared	Factor
1	SIDH-p434-compressed	1	197	1	197	1.5	28	1	110	6.9
2	SIKE-p434-compressed	1	197	1	236	1.8	350	12.5	16	1
3	SIDH-p503-compressed	2	225	1.1	225	1.8	32	1.1	126	7.9
4	SIKE-p503-compressed	2	225	1.1	280	2.2	407	14.5	24	1.5
5	SIDH-p610-compressed	3	274	1.4	274	2.1	39	1.4	154	9.6
6	SIKE-p610-compressed	3	274	1.4	336	2.6	491	17.5	24	1.5
7	SIDH-p434	1	330	1.7	330	2.6	28	1	110	6.9
8	SIKE-p434	1	330	1.7	346	2.7	374	13.4	16	1
9	SIDH-p751-compressed	5	335	1.7	335	2.6	48	1.7	188	11.8
10	SIKE-p751-compressed	5	335	1.7	410	3.2	602	21.5	32	2
11	SIDH-p503	2	378	1.9	378	3	32	1.1	126	7.9
12	SIKE-p503	2	378	1.9	402	3.1	434	15.5	24	1.5
13	SIDH-p610	3	462	2.3	462	3.6	39	1.4	154	9.6
14	SIKE-p610	3	462	2.3	486	3.8	524	18.7	24	1.5
15	SIDH-p751	5	564	2.9	564	4.4	48	1.7	188	11.8
16	SIKE-p751	5	564	2.9	596	4.7	644	23	32	2
17	LightSaber-KEM	1	672	3.4	736	5.8	1568	56	32	2
18	NTRU-HPS-2048-509	1	699	3.5	699	5.5	935	33.4	32	2
19	Kyber512	1	800	4.1	768	6	1632	58.3	32	2
20	Kyber512-90s	1	800	4.1	768	6	1632	58.3	32	2
21	ntrulpr653	1	897	4.6	1025	8	1125	40.2	32	2
22	NTRU-HPS-2048-677	3	930	4.7	930	7.3	1234	44.1	32	2
23	Saber-KEM	3	992	5	1088	8.5	2304	82.3	32	2
24	sntrup653	1	994	5	897	7	1518	54.2	32	2
25	ntrulpr761	2	1039	5.3	1167	9.1	1294	46.2	32	2
26	NTRU-HRSS-701	3	1138	5.8	1138	8.9	1450	51.8	32	2
27	sntrup761	2	1158	5.9	1039	8.1	1763	63	32	2
28	Kyber768	3	1184	6	1088	8.5	2400	85.7	32	2
29	Kyber768-90s	3	1184	6	1088	8.5	2400	85.7	32	2
30	ntrulpr857	3	1184	6	1312	10.3	1463	52.3	32	2
29	FrodoKEM-640-SHAKE	Lattice (LWE)	3148720	103.7	4267582	131.8	4093162	214.8	3	3	3	9
30	FrodoKEM-1344-AES	Lattice (LWE)	4664746	153.6	5829221	180	5379988	282.3	3	3	3	9

1	Kyber512-90s	Lattice (LWE)	30									
2	Kyber512	Lattice (LWE)	30									
3	Kyber768-90s	Lattice (LWE)	30									
4	Kyber1024-90s	Lattice (LWE)	28									
5	LightSaber-KEM	Lattice (LWE+R)	28									
6	NTRU-HPS-2048-509	Lattice	28									
7	Kyber768	Lattice (LWE)	26									
8	NTRU-HRSS-701	Lattice	26									
9	Kyber1024	Lattice (LWE)	24									
10	ntrulpr653	Lattice	24									
11	ntrulpr761	Lattice	24									
12	Saber-KEM	Lattice (LWE+R)	24									
13	ntrulpr857	Lattice	24									
14	FireSaber-KEM	Lattice (LWE+R)	24									
15	ntrulpr1277	Lattice	24									
16	NTRU-HPS-2048-677	Lattice	24									
17	HQC-128	Code-based	21									
18	NTRU-HPS-4096-821	Lattice	21									
19	sntrup653	Lattice	21									
20	sntrup761	Lattice	21									
21	sntrup857	Lattice	21									
22	sntrup1277	Lattice	21									
23	Classic-McEliece-348864f	Code-based	18									
24	BIKE-L1	Code-based	16									
25	BIKE-L3	Code-based	16									
26	Classic-McEliece-348864	Code-based	16									
27	HQC-192	Code-based	15									
28	FrodoKEM-640-AES	Lattice (LWE)	15									
29	HQC-256	Code-based	13									
30	NTRU-HPS-4096-1229	Lattice	13									
31	NTRU-HRSS-1373	Lattice	13									
32	Classic-McEliece-6688128	Code-based	13									

31	SIDH-p434	Isogeny	5654091	186.1	11647942	359.7	4520865	237.2	3	3	3	9
32	SIKE-p434	Isogeny	6197253	204	10097567	311.8	10904398	572.2	3	3	3	9
33	FrodoKEM-976-SHAKE	Lattice (LWE)	7021826	231.2	9243416	285.4	9109536	478	3	3	3	9
34	SIDH-p503	Isogeny	7956605	261.9	15930604	491.9	6316223	331.5	3	3	3	9
35	SIKE-p503	Isogeny	8671491	285.5	14285437	441.1	15090345	791.9	3	3	3	9
36	SIKE-p434-compressed	Isogeny	11151196	367.1	16091778	496.9	11552400	606.2	3	3	3	9
37	SIDH-p434-compressed	Isogeny	11209821	369	16467316	508.5	5101392	267.7	3	3	3	9
38	NTRU-HPS-4096-1229	Lattice	12406657	408.4	460803	14.2	741261	38.9	3	5	5	13
39	FrodoKEM-1344-SHAKE	Lattice (LWE)	12601112	414.8	16107704	497.4	15400715	808.2	3	3	3	9
40	SIKE-p503-compressed	Isogeny	15026278	494.7	22251822	687.1	15891889	834	3	3	3	9
41	SIDH-p503-compressed	Isogeny	15242166	501.8	22995082	710	7088573	372	3	3	3	9
42	NTRU-HRSS-1373	Lattice	15922068	524.1	340073	10.5	895579	47	3	5	5	13
43	SIKE-p610	Isogeny	16516384	543.7	30973808	956.4	30794480	1616	3	3	0	6
44	SIDH-p610	Isogeny	17128219	563.9	32146290	992.6	13851706	726.9	3	3	3	9
45	SIDH-p751	Isogeny	23617554	777.5	48813951	1507.3	19233242	1009.3	3	0	0	3
46	SIKE-p751	Isogeny	26485988	871.9	42886151	1324.2	47206045	2477.2	3	0	0	3
47	SIDH-p610-compressed	Isogeny	30775700	1013.1	44161525	1363.6	15152561	795.2	0	0	3	3
48	SIKE-p610-compressed	Isogeny	33401019	1099.5	42572480	1314.5	32317016	1695.9	0	0	0	0
49	SIKE-p751-compressed	Isogeny	44187613	1454.6	68414496	2112.5	49102481	2576.7	0	0	0	0
50	SIDH-p751-compressed	Isogeny	44403930	1461.8	67441010	2082.4	21077729	1106.1	0	0	0	0
51	Classic-McEliece-348864f	Code-based	349054107	11490.7	61805	1.9	184433	9.7	0	10	8	18
52	Classic-McEliece-348864	Code-based	472549759	15556.2	81172	2.5	188381	9.9	0	8	8	16
53	Classic-McEliece-6688128	Code-based	899451646	29609.6	228951	7.1	529520	27.8	0	8	5	13
54	Classic-McEliece-460896f	Code-based	1109936192	36538.7	140594	4.3	430599	22.6	0	8	5	13
55	Classic-McEliece-6960119f	Code-based	1323860114	43581	187544	5.8	473180	24.8	0	8	5	13
56	Classic-McEliece-6688128f	Code-based	1400749128	46112.2	206238	6.4	511853	26.9	0	8	5	13
57	Classic-McEliece-8192128f	Code-based	1440024054	47405.1	202961	6.3	520936	27.3	0	8	5	13
58	Classic-McEliece-460896	Code-based	1496892143	49277.2	146998	4.5	426035	22.4	0	8	5	13
59	Classic-McEliece-6960119	Code-based	1587651773	52264.9	187845	5.8	464536	24.4	0	8	5	13
60	Classic-McEliece-8192128	Code-based	2654019753	87369.4	203049	6.3	526242	27.6	0	8	5	13

https://asecuritysite.com/pqc/

KEM/Public Key Encryption

		Level	Public	Factor	Cipher	Factor	Secret	Factor	Shared	Factor
1	SIDH-p434-compressed	1	197	1	197	1.5	28	1	110	6.9
2	SIKE-p434-compressed	1	197	1	236	1.8	350	12.5	16	1
3	SIDH-p503-compressed	2	225	1.1	225	1.8	32	1.1	126	7.9
4	SIKE-p503-compressed	2	225	1.1	280	2.2	407	14.5	24	1.5
5	SIDH-p610-compressed	3	274	1.4	274	2.1	39	1.4	154	9.6
6	SIKE-p610-compressed	3	274	1.4	336	2.6	491	17.5	24	1.5
7	SIDH-p434	1	330	1.7	330	2.6	28	1	110	6.9
8	SIKE-p434	1	330	1.7	346	2.7	374	13.4	16	1
9	SIDH-p751-compressed	5	335	1.7	335	2.6	48	1.7	188	11.8
10	SIKE-p751-compressed	5	335	1.7	410	3.2	602	21.5	32	2
11	SIDH-p503	2	378	1.9	378	3	32	1.1	126	7.9
12	SIKE-p503	2	378	1.9	402	3.1	434	15.5	24	1.5
13	SIDH-p610	3	462	2.3	462	3.6	39	1.4	154	9.6
14	SIKE-p610	3	462	2.3	486	3.8	524	18.7	24	1.5
15	SIDH-p751	5	564	2.9	564	4.4	48	1.7	188	11.8
16	SIKE-p751	5	564	2.9	596	4.7	644	23	32	2
17	LightSaber-KEM	1	672	3.4	736	5.8	1568	56	32	2
18	NTRU-HPS-2048-509	1	699	3.5	699	5.5	935	33.4	32	2
19	Kyber512	1	800	4.1	768	6	1632	58.3	32	2
20	Kyber512-90s	1	800	4.1	768	6	1632	58.3	32	2
21	ntrulpr653	1	897	4.6	1025	8	1125	40.2	32	2
22	NTRU-HPS-2048-677	3	930	4.7	930	7.3	1234	44.1	32	2
23	Saber-KEM	3	992	5	1088	8.5	2304	82.3	32	2
24	sntrup653	1	994	5	897	7	1518	54.2	32	2
25	ntrulpr761	2	1039	5.3	1167	9.1	1294	46.2	32	2
26	NTRU-HRSS-701	3	1138	5.8	1138	8.9	1450	51.8	32	2
27	sntrup761	2	1158	5.9	1039	8.1	1763	63	32	2
28	Kyber768	3	1184	6	1088	8.5	2400	85.7	32	2
29	Kyber768-90s	3	1184	6	1088	8.5	2400	85.7	32	2
30	ntrulpr857	3	1184	6	1312	10.3	1463	52.3	32	2
29	FrodoKEM-640-SHAKE	Lattice (LWE)	3148720	103.7	4267582	131.8	4093162	214.8	3	3	3	9
30	FrodoKEM-1344-AES	Lattice (LWE)	4664746	153.6	5829221	180	5379988	282.3	3	3	3	9

1	Kyber512-90s	Lattice (LWE)	30
2	Kyber512	Lattice (LWE)	30
3	Kyber768-90s	Lattice (LWE)	30
4	Kyber1024-90s	Lattice (LWE)	28
5	LightSaber-KEM	Lattice (LWE+R)	28
6	NTRU-HPS-2048-509	Lattice	28
7	Kyber768	Lattice (LWE)	26
8	NTRU-HRSS-701	Lattice	26
9	Kyber1024	Lattice (LWE)	24
10	ntrulpr653	Lattice	24
11	ntrulpr761	Lattice	24
12	Saber-KEM	Lattice (LWE+R)	24
13	ntrulpr857	Lattice	24
14	FireSaber-KEM	Lattice (LWE+R)	24
15	ntrulpr1277	Lattice	24
16	NTRU-HPS-2048-677	Lattice	24
17	HQC-128	Code-based	21
18	NTRU-HPS-4096-821	Lattice	21
19	sntrup653	Lattice	21
20	sntrup761	Lattice	21
21	sntrup857	Lattice	21
22	sntrup1277	Lattice	21
23	Classic-McEliece-348864f	Code-based	18
24	BIKE-L1	Code-based	16
25	BIKE-L3	Code-based	16
26	Classic-McEliece-348864	Code-based	16
27	HQC-192	Code-based	15
28	FrodoKEM-640-AES	Lattice (LWE)	15
29	HQC-256	Code-based	13
30	NTRU-HPS-4096-1229	Lattice	13
31	NTRU-HRSS-1373	Lattice	13
32	Classic-McEliece-6688128	Code-based	13

31	NTRU-HPS-4096-821	5	1230	6.2	1230	9.6	1590	56.8	32	2
32	FireSaber-KEM	5	1312	6.7	1472	11.5	3040	108.6	32	2
33	sntrup857	3	1322	6.7	1184	9.3	1999	71.4	32	2
34	BIKE-L1	1	1541	7.8	1573	12.3	5223	186.5	32	2
35	Kyber1024	5	1568	8	1568	12.3	3168	113.1	32	2
36	Kyber1024-90s	5	1568	8	1568	12.3	3168	113.1	32	2
37	NTRU-HPS-4096-1229	5	1842	9.4	1842	14.4	2366	84.5	32	2
38	ntrulpr1277	5	1847	9.4	1975	15.4	2231	79.7	32	2
39	sntrup1277	5	2067	10.5	1847	14.4	3059	109.3	32	2
40	HQC-128	1	2249	11.4	4481	35	2289	81.8	64	4
41	NTRU-HRSS-1373	5	2401	12.2	2401	18.8	2983	106.5	32	2
42	BIKE-L3	3	3083	15.6	3115	24.3	10105	360.9	32	2
43	HQC-192	3	4522	23	9026	70.5	4562	162.9	64	4
44	HQC-256	5	7245	36.8	14469	113	7285	260.2	64	4
45	FrodoKEM-640-AES	1	9616	48.8	9720	75.9	19888	710.3	16	1
46	FrodoKEM-640-SHAKE	1	9616	48.8	9720	75.9	19888	710.3	16	1
47	FrodoKEM-976-AES	3	15632	79.4	15744	123	31296	1117.7	24	1.5
48	FrodoKEM-976-SHAKE	3	15632	79.4	15744	123	31296	1117.7	24	1.5
49	FrodoKEM-1344-AES	5	21520	109.2	21632	169	43088	1538.9	32	2
50	FrodoKEM-1344-SHAKE	5	21520	109.2	21632	169	43088	1538.9	32	2
51	Classic-McEliece-348864	1	261120	1325.5	128	1	6452	230.4	32	2
52	Classic-McEliece-348864f	1	261120	1325.5	128	1	6452	230.4	32	2
53	Classic-McEliece-460896	3	524160	2660.7	188	1.5	13568	484.6	32	2
54	Classic-McEliece-460896f	3	524160	2660.7	188	1.5	13568	484.6	32	2
55	Classic-McEliece-6688128	5	1044992	5304.5	240	1.9	13892	496.1	32	2
56	Classic-McEliece-6688128f	5	1044992	5304.5	240	1.9	13892	496.1	32	2
57	Classic-McEliece-6960119	5	1047319	5316.3	226	1.8	13908	496.7	32	2
58	Classic-McEliece-6960119f	5	1047319	5316.3	226	1.8	13908	496.7	32	2
59	Classic-McEliece-8192128	5	1357824	6892.5	240	1.9	14080	502.9	32	2
60	Classic-McEliece-8192128f	5	1357824	6892.5	240	1.9	14080	502.9	32	2

https://asecuritysite.com/pqc/

Digital Signatures

	Type	Method	Key gen	Factor	Sign	Factor	Verify	Score (Dec)	Overall	Score (Enc)	Score (Dec)	Overall
1	picnic3_L1	Hash/ZKP	21187	1	23348413	100.7	19186836	261.3	10	3	3	16
2	picnic_L3_full	Hash/ZKP	22571	1.1	10877666	46.9	12483100	170	10	5	5	20
3	picnic3_L3	Hash/ZKP	24184	1.1	47229332	203.7	36701854	499.9	10	3	5	18
4	picnic3_L5	Hash/ZKP	25035	1.2	75838669	327.1	58815102	801	10	3	5	18
5	picnic_L1_full	Hash/ZKP	25052	1.2	4854620	20.9	5917140	80.6	10	5	8	23
6	picnic_L5_full	Hash/ZKP	25414	1.2	15670149	67.6	13711927	186.7	10	5	5	20
7	picnic_L1_UR	Hash/ZKP	26617	1.3	9290846	40.1	7755920	105.6	10	5	5	20
8	picnic_L1_FS	Hash/ZKP	26839	1.3	8025732	34.6	6777198	92.3	10	5	8	23
9	picnic_L3_UR	Hash/ZKP	36216	1.7	23156076	99.9	21778246	296.6	10	5	5	20
10	picnic_L5_UR	Hash/ZKP	41418	2	39313637	169.5	34446115	469.1	8	3	5	16
11	picnic_L5_FS	Hash/ZKP	44942	2.1	37066573	159.9	28836125	392.7	8	3	5	16
12	Dilithium2-AES	Lattice	67697	3.2	231878	1	73424	1	8	10	10	28
13	picnic_L3_FS	Hash/ZKP	72919	3.4	20939121	90.3	18323702	249.6	8	5	3	16
14	Dilithium3-AES	Lattice	104515	4.9	374297	1.6	113536	1.5	8	10	10	28
15	Dilithium2	Lattice	116511	5.5	342726	1.5	112506	1.5	8	10	10	28
16	Dilithium5-AES	Lattice	164148	7.7	416647	1.8	166482	2.3	8	10	8	26
17	Dilithium3	Lattice	191331	9	534254	2.3	180350	2.5	8	8	8	24
18	Dilithium5	Lattice	307765	14.5	610807	2.6	417971	5.7	5	8	8	21
19	SPHINCS+-Haraka-128f-s	Hash	1114859	52.6	27587176	119	1521957	20.7	5	3	5	13
20	SPHINCS+-Haraka-128f-r	Hash	1296477	61.2	31847101	137.3	2308807	31.4	5	3	5	13
21	SPHINCS+-Haraka-192f-s	Hash	1733557	81.8	47717755	205.8	2509526	34.2	5	3	5	13
22	SPHINCS+-Haraka-192f-r	Hash	2034468	96	61820381	266.6	3691384	50.3	5	3	5	13
23	SPHINCS+-SHA256-128f-s	Hash	3088404	145.8	72191077	311.3	8962488	122.1	3	3	3	9
24	SPHINCS+-SHA256-192f-s	Hash	3920103	185	119085653	513.6	12269960	167.1	3	3	3	9
25	SPHINCS+-SHAKE256-128f-s	Hash	3993469	188.5	118778065	512.2	11837565	161.2	3	3	3	9
26	SPHINCS+-SHA256-128f-r	Hash	4576725	216	115180200	496.7	16236483	221.1	3	3	3	9
27	SPHINCS+-Haraka-256f-s	Hash	4656137	219.8	89990034	388.1	2534462	34.5	3	3	5	11
28	SPHINCS+-SHAKE256-192f-s	Hash	5766316	272.2	166899175	719.8	16662894	226.9	3	3	3	9
29	SPHINCS+-SHA256-192f-r	Hash	6343609	299.4	187556226	808.9	22966293	312.8	3	3	3	9
30	SPHINCS+-Haraka-256f-r	Hash	6398014	302	119210092	514.1	3793534	51.7	3	3	5	11
31	SPHINCS+-SHAKE256-128f-r	Hash	6831602	322.4	176444474	760.9	21769720	296.5	3	3	3	9
32	SPHINCS+-SHAKE256-192f-r	Hash	9735939	459.5	284106213	1225.2	30835025	420	3	0	3	6
33	SPHINCS+-SHA256-256f-s	Hash	10771651	508.4	220637782	951.5	12168947	165.7	3	3	3	9
34	SPHINCS+-SHAKE256-256f-s	Hash	15684219	740.3	318508169	1373.6	16422940	223.7	3	0	3	6
35	Falcon-512	Lattice	24656358	1163.7	1085984	4.7	183949	2.5	0	8	8	16
36	SPHINCS+-SHAKE256-256f-r	Hash	27044805	1276.5	564867404	2436.1	32709637	445.5	0	0	3	3
37	SPHINCS+-SHA256-256f-r	Hash	28940978	1366	571268028	2463.7	29935214	407.7	0	0	3	3
38	SPHINCS+-Haraka-256s-s	Hash	69028778	3258.1	998765490	4307.3	1330020	18.1	0	0	5	5
39	Falcon-1024	Lattice	74741342	3527.7	2204927	9.5	359553	4.9	0	8	8	16
40	SPHINCS+-Haraka-128s-r	Hash	78280311	3694.7	622976746	2686.7	1634310	22.3	0	0	5	5
41	SPHINCS+-Haraka-128s-s	Hash	79319904	3743.8	573371215	2472.7	665009	9.1	0	0	8	8
42	SPHINCS+-Haraka-256s-r	Hash	97499330	4601.8	1332794011	5747.8	2003259	27.3	0	0	5	5
43	Rainbow-V-Circumzenithal	MQ	104248763	4920.4	130818975	564.2	147868082	2013.9	3	0	3	0
44	SPHINCS+-Haraka-192s-s	Hash	107353443	5066.9	1055976583	4554	846444	11.5	0	0	5	5
45	SPHINCS+-Haraka-192s-r	Hash	122640669	5788.5	1322115681	5701.8	1335915	18.2	0	0	5	5
46	SPHINCS+-SHA256-256s-s	Hash	164581875	7768.1	2072066784	8936	5872652	80	0	0	5	5
47	SPHINCS+-SHA256-128s-s	Hash	177182865	8362.8	1487638276	6415.6	2787373	38	0	0	5	5
48	Rainbow-V-Compressed	MQ	200893415	9481.9	1728316371	7453.6	150730999	2052.9	3	0	0	0
49	SPHINCS+-SHAKE256-128s-s	Hash	245590268	11591.6	1847092410	7965.8	3595958	49	0	0	5	5
50	SPHINCS+-SHA256-192s-s	Hash	253977643	11987.4	2457142599	10596.7	4148624	56.5	0	0	5	5
51	SPHINCS+-SHAKE256-256s-s	Hash	255886895	12077.5	3085396286	13306.1	8930086	121.6	0	0	3	3
52	SPHINCS+-SHA256-128s-r	Hash	280035862	13217.3	2271055091	9794.2	5982596	81.5	0	0	5	5
53	SPHINCS+-SHAKE256-192s-s	Hash	360685288	17023.9	3486117790	15034.3	5642190	76.8	0	0	5	5
54	SPHINCS+-SHA256-192s-r	Hash	397446085	18759	3710077050	16000.1	8174834	111.3	0	0	3	3
55	SPHINCS+-SHAKE256-256s-r	Hash	428926274	20244.8	664184504	2864.4	16498665	224.7	0	0	3	3
56	SPHINCS+-SHAKE256-128s-r	Hash	446377018	21068.4	3274888436	14123.3	6274742	85.5	0	0	5	5
57	Rainbow-I-Classic	MQ	492275357	23234.8	6220698	26.8	6573578	89.5	0	5	5	10
58	Rainbow-I-Compressed	MQ	542931029	25625.7	235804438	1016.9	7520590	102.4	0	0	3	3
59	Rainbow-I-Circumzenithal	MQ	544285874	25689.6	6271496	27	7651253	104.2	0	5	3	8
60	SPHINCS+-SHA256-256s-r	Hash	635319074	29986.3	1889802242	8150	15630510	212.9	0	0	3	3
61	SPHINCS+-SHAKE256-192s-r	Hash	661757883	31234.1	1701951015	7339.9	10728453	146.1	0	0	3	3
62	Rainbow-V-Classic	MQ	1707296542	80582.3	132498942	571.4	139224699	1896.2	0	3	0	3
63	Rainbow-III-Classic	MQ	2306828504	108879.4	59391493	256.1	61864256	842.6	0	3	3	6
64	Rainbow-III-Circumzenithal	MQ	3264803255	154094.6	59377096	256.1	70084535	954.5	0	3	3	6
65	Rainbow-III-Compressed	MQ	3292975486	155424.3	3593929070	15499.2	75246328	1024.8	0	0	0	0

1	Dilithium2-AES	28
2	Dilithium3-AES	28
3	Dilithium2	28
4	Dilithium5-AES	26
5	Dilithium3	24
6	picnic_L1_full	23
7	picnic_L1_FS	23
8	Dilithium5	21
9	picnic_L3_full	20
10	picnic_L5_full	20
11	picnic_L1_UR	20
12	picnic_L3_UR	20
13	picnic3_L3	18
14	picnic3_L5	18
15	picnic3_L1	16
16	picnic_L5_UR	16
17	picnic_L5_FS	16
18	picnic_L3_FS	16
19	Falcon-512	16
20	Falcon-1024	16
21	SPHINCS+-Haraka-12	13
22	SPHINCS+-Haraka-12	13
23	SPHINCS+-Haraka-19	13
24	SPHINCS+-Haraka-19	13
25	SPHINCS+-Haraka-25	11
26	SPHINCS+-Haraka-25	11
27	Rainbow-I-Classic	10
28	SPHINCS+-SHA256-12	9
29	SPHINCS+-SHA256-19	9

https://asecuritysite.com/pqc/

Digital Signatures

	Method	Level	Pub key(B)	Factor	Pri key (B)	Factor	Sig (B)	Factor
1	SPHINCS+-Haraka-128f-robust	1	32	1	64	1.3	17088	258.9
2	SPHINCS+-Haraka-128f-simple	1	32	1	64	1.3	17088	258.9
3	SPHINCS+-Haraka-128s-robust	1	32	1	64	1.3	7856	119
4	SPHINCS+-Haraka-128s-simple	1	32	1	64	1.3	7856	119
5	SPHINCS+-SHA256-128f-robust	1	32	1	64	1.3	17088	258.9
6	SPHINCS+-SHA256-128f-simple	1	32	1	64	1.3	17088	258.9
7	SPHINCS+-SHA256-128s-robust	1	32	1	64	1.3	7856	119
8	SPHINCS+-SHA256-128s-simple	1	32	1	64	1.3	7856	119
9	SPHINCS+-SHAKE256-128f-robust	1	32	1	64	1.3	17088	258.9
10	SPHINCS+-SHAKE256-128f-simple	1	32	1	64	1.3	17088	258.9
11	SPHINCS+-SHAKE256-128s-robust	1	32	1	64	1.3	7856	119
12	SPHINCS+-SHAKE256-128s-simple	1	32	1	64	1.3	7856	119
13	picnic_L1_FS	1	33	1	49	1	34036	515.7
14	picnic_L1_UR	1	33	1	49	1	53965	817.7
15	picnic_L1_full	1	35	1.1	52	1.1	32065	485.8
16	picnic3_L1	1	35	1.1	52	1.1	14612	221.4
17	SPHINCS+-Haraka-192f-robust	3	48	1.5	96	2	35664	540.4
18	SPHINCS+-Haraka-192f-simple	3	48	1.5	96	2	35664	540.4
19	SPHINCS+-Haraka-192s-robust	3	48	1.5	96	2	16224	245.8
20	SPHINCS+-Haraka-192s-simple	3	48	1.5	96	2	16224	245.8
21	SPHINCS+-SHA256-192f-robust	3	48	1.5	96	2	35664	540.4
22	SPHINCS+-SHA256-192f-simple	3	48	1.5	96	2	35664	540.4
23	SPHINCS+-SHA256-192s-robust	3	48	1.5	96	2	16224	245.8
24	SPHINCS+-SHA256-192s-simple	3	48	1.5	96	2	16224	245.8
25	SPHINCS+-SHAKE256-192f-robust	3	48	1.5	96	2	35664	540.4
26	SPHINCS+-SHAKE256-192f-simple	3	48	1.5	96	2	35664	540.4
27	SPHINCS+-SHAKE256-192s-robust	3	48	1.5	96	2	16224	245.8
28	SPHINCS+-SHAKE256-192s-simple	3	48	1.5	96	2	16224	245.8
29	picnic_L3_FS	3	49	1.5	73	1.5	76776	1163.3
30	picnic_L3_UR	3	49	1.5	73	1.5	121849	1846.2

29	SPHINCS+-SHA256-192f-r	Hash	6343609	299.4	187556226	808.9	22966293	312.8	3	3	3	9
30	SPHINCS+-Haraka-256f-r	Hash	6398014	302	119210092	514.1	3793534	51.7	3	3	5	11
31	SPHINCS+-SHAKE256-128f-r	Hash	6831602	322.4	176444474	760.9	21769720	296.5	3	3	3	9
32	SPHINCS+-SHAKE256-192f-r	Hash	9735939	459.5	284106213	1225.2	30835025	420	3	0	3	6
33	SPHINCS+-SHA256-256f-s	Hash	10771651	508.4	220637782	951.5	12168947	165.7	3	3	3	9
34	SPHINCS+-SHAKE256-256f-s	Hash	15684219	740.3	318508169	1373.6	16422940	223.7	3	0	3	6
35	Falcon-512	Lattice	24656358	1163.7	1085984	4.7	183949	2.5	0	8	8	16
36	SPHINCS+-SHAKE256-256f-r	Hash	27044805	1276.5	564867404	2436.1	32709637	445.5	0	0	3	3
37	SPHINCS+-SHA256-256f-r	Hash	28940978	1366	571268028	2463.7	29935214	407.7	0	0	3	3
38	SPHINCS+-Haraka-256s-s	Hash	69028778	3258.1	998765490	4307.3	1330020	18.1	0	0	5	5
39	Falcon-1024	Lattice	74741342	3527.7	2204927	9.5	359553	4.9	0	8	8	16
40	SPHINCS+-Haraka-128s-r	Hash	78280311	3694.7	622976746	2686.7	1634310	22.3	0	0	5	5
41	SPHINCS+-Haraka-128s-s	Hash	79319904	3743.8	573371215	2472.7	665009	9.1	0	0	8	8
42	SPHINCS+-Haraka-256s-r	Hash	97499330	4601.8	1332794011	5747.8	2003259	27.3	0	0	5	5
43	Rainbow-V-Circumzenithal	MQ	104248763	4920.4	130818975	564.2	147868082	2013.9	3	0	3	0
44	SPHINCS+-Haraka-192s-s	Hash	107353443	5066.9	1055976583	4554	846444	11.5	0	0	5	5
45	SPHINCS+-Haraka-192s-r	Hash	122640669	5788.5	1322115681	5701.8	1335915	18.2	0	0	5	5
46	SPHINCS+-SHA256-256s-s	Hash	164581875	7768.1	2072066784	8936	5872652	80	0	0	5	5
47	SPHINCS+-SHA256-128s-s	Hash	177182865	8362.8	1487638276	6415.6	2787373	38	0	0	5	5
48	Rainbow-V-Compressed	MQ	200893415	9481.9	1728316371	7453.6	150730999	2052.9	3	0	0	0
49	SPHINCS+-SHAKE256-128s-s	Hash	245590268	11591.6	1847092410	7965.8	3595958	49	0	0	5	5
50	SPHINCS+-SHA256-192s-s	Hash	253977643	11987.4	2457142599	10596.7	4148624	56.5	0	0	5	5
51	SPHINCS+-SHAKE256-256s-s	Hash	255886895	12077.5	3085396286	13306.1	8930086	121.6	0	0	3	3
52	SPHINCS+-SHA256-128s-r	Hash	280035862	13217.3	2271055091	9794.2	5982596	81.5	0	0	5	5
53	SPHINCS+-SHAKE256-192s-s	Hash	360685288	17023.9	3486117790	15034.3	5642190	76.8	0	0	5	5
54	SPHINCS+-SHA256-192s-r	Hash	397446085	18759	3710077050	16000.1	8174834	111.3	0	0	3	3
55	SPHINCS+-SHAKE256-256s-r	Hash	428926274	20244.8	664184504	2864.4	16498665	224.7	0	0	3	3

56	SPHINCS+-SHAKE256-128s-r	Hash	446377018	21068.4	3274888436	14123.3	6274742	85.5	0	0	5	5
57	Rainbow-I-Classic	MQ	492275357	23234.8	6220698	26.8	6573578	89.5	0	5	5	10
58	Rainbow-I-Compressed	MQ	542931029	25625.7	235804438	1016.9	7520590	102.4	0	0	3	3
59	Rainbow-I-Circumzenithal	MQ	544285874	25689.6	6271496	27	7651253	104.2	0	5	3	8
60	SPHINCS+-SHA256-256s-r	Hash	635319074	29986.3	1889802242	8150	15630510	212.9	0	0	3	3
61	SPHINCS+-SHAKE256-192s-r	Hash	661757883	31234.1	1701951015	7339.9	10728453	146.1	0	0	3	3
62	Rainbow-V-Classic	MQ	1707296542	80582.3	132498942	571.4	139224699	1896.2	0	3	0	3
63	Rainbow-III-Classic	MQ	2306828504	108879.4	59391493	256.1	61864256	842.6	0	3	3	6
64	Rainbow-III-Circumzenithal	MQ	3264803255	154094.6	59377096	256.1	70084535	954.5	0	3	3	6
65	Rainbow-III-Compressed	MQ	3292975486	155424.3	3593929070	15499.2	75246328	1024.8	0	0	0	0

1	Dilithium2-AES	28
2	Dilithium3-AES	28
3	Dilithium2	28
4	Dilithium5-AES	26
5	Dilithium3	24
6	picnic_L1_full	23
7	picnic_L1_FS	23
8	Dilithium5	21
9	picnic_L3_full	20
10	picnic_L5_full	20
11	picnic_L1_UR	20
12	picnic_L3_UR	20
13	picnic3_L3	18
14	picnic3_L5	18
15	picnic3_L1	16
16	picnic_L5_UR	16
17	picnic_L5_FS	16
18	picnic_L3_FS	16
19	Falcon-512	16
20	Falcon-1024	16
21	SPHINCS+-Haraka-12	13
22	SPHINCS+-Haraka-12	13
23	SPHINCS+-Haraka-19	13
24	SPHINCS+-Haraka-19	13
25	SPHINCS+-Haraka-25	11
26	SPHINCS+-Haraka-25	11
27	Rainbow-I-Classic	10
28	SPHINCS+-SHA256-12	9
29	SPHINCS+-SHA256-19	9

https://asecuritysite.com/pqc/

Digital Signatures

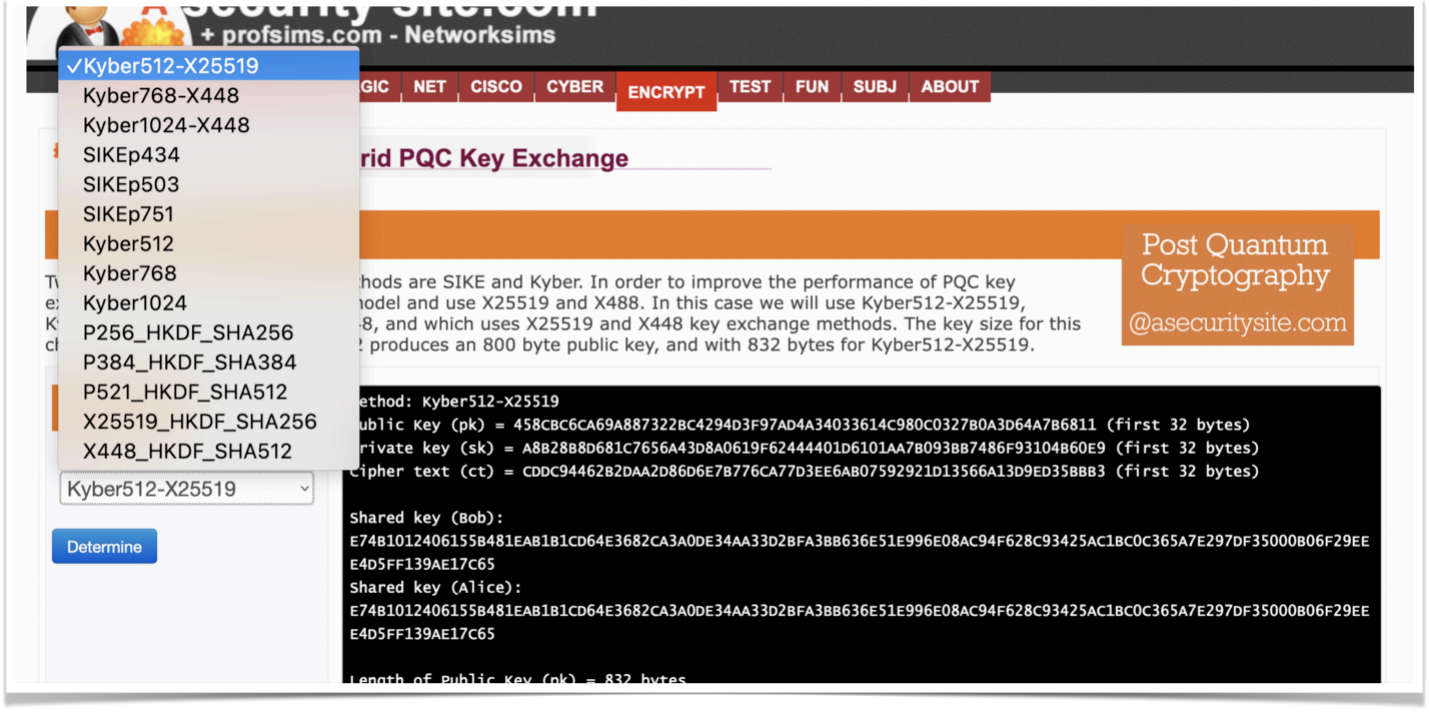
	Method	Level	Pub key(B)	Factor	Pri key (B)	Factor	Sig (B)	Factor
1	SPHINCS+-Haraka-128f-robust	1	32	1	64	1.3	17088	258.9
2	SPHINCS+-Haraka-128f-simple	1	32	1	64	1.3	17088	258.9
3	SPHINCS+-Haraka-128s-robust	1	32	1	64	1.3	7856	119
4	SPHINCS+-Haraka-128s-simple	1	32	1	64	1.3	7856	119
5	SPHINCS+-SHA256-128f-robust	1	32	1	64	1.3	17088	258.9
6	SPHINCS+-SHA256-128f-simple	1	32	1	64	1.3	17088	258.9
7	SPHINCS+-SHA256-128s-robust	1	32	1	64	1.3	7856	119
8	SPHINCS+-SHA256-128s-simple	1	32	1	64	1.3	7856	119
9	SPHINCS+-SHAKE256-128f-robust	1	32	1	64	1.3	17088	258.9
10	SPHINCS+-SHAKE256-128f-simple	1	32	1	64	1.3	17088	258.9
11	SPHINCS+-SHAKE256-128s-robust	1	32	1	64	1.3	7856	119
12	SPHINCS+-SHAKE256-128s-simple	1	32	1	64	1.3	7856	119
13	picnic_L1_FS	1	33	1	49	1	34036	515.7
14	picnic_L1_UR	1	33	1	49	1	53965	817.7
15	picnic_L1_full	1	35	1.1	52	1.1	32065	485.8
16	picnic3_L1	1	35	1.1	52	1.1	14612	221.4
17	SPHINCS+-Haraka-192f-robust	3	48	1.5	96	2	35664	540.4
18	SPHINCS+-Haraka-192f-simple	3	48	1.5	96	2	35664	540.4
19	SPHINCS+-Haraka-192s-robust	3	48	1.5	96	2	16224	245.8
20	SPHINCS+-Haraka-192s-simple	3	48	1.5	96	2	16224	245.8
21	SPHINCS+-SHA256-192f-robust	3	48	1.5	96	2	35664	540.4
22	SPHINCS+-SHA256-192f-simple	3	48	1.5	96	2	35664	540.4
23	SPHINCS+-SHA256-192s-robust	3	48	1.5	96	2	16224	245.8
24	SPHINCS+-SHA256-192s-simple	3	48	1.5	96	2	16224	245.8
25	SPHINCS+-SHAKE256-192f-robust	3	48	1.5	96	2	35664	540.4
26	SPHINCS+-SHAKE256-192f-simple	3	48	1.5	96	2	35664	540.4
27	SPHINCS+-SHAKE256-192s-robust	3	48	1.5	96	2	16224	245.8
28	SPHINCS+-SHAKE256-192s-simple	3	48	1.5	96	2	16224	245.8
29	picnic_L3_FS	3	49	1.5	73	1.5	76776	1163.3
30	picnic_L3_UR	3	49	1.5	73	1.5	121849	1846.2
31	picnic_L3_full	3	49	1.5	73	1.5	71183	1078.5
32	picnic3_L3	3	49	1.5	73	1.5	35028	530.7
33	SPHINCS+-Haraka-256f-robust	5	64	2	128	2.6	49856	755.4
34	SPHINCS+-Haraka-256f-simple	5	64	2	128	2.6	49856	755.4
35	SPHINCS+-Haraka-256s-robust	5	64	2	128	2.6	29792	451.4
36	SPHINCS+-Haraka-256s-simple	5	64	2	128	2.6	29792	451.4
37	SPHINCS+-SHA256-256f-robust	5	64	2	128	2.6	49856	755.4
38	SPHINCS+-SHA256-256f-simple	5	64	2	128	2.6	49856	755.4
39	SPHINCS+-SHA256-256s-robust	5	64	2	128	2.6	29792	451.4
40	SPHINCS+-SHA256-256s-simple	5	64	2	128	2.6	29792	451.4
41	SPHINCS+-SHAKE256-256f-robust	5	64	2	128	2.6	49856	755.4
42	SPHINCS+-SHAKE256-256f-simple	5	64	2	128	2.6	49856	755.4
43	SPHINCS+-SHAKE256-256s-robust	5	64	2	128	2.6	29792	451.4
44	SPHINCS+-SHAKE256-256s-simple	5	64	2	128	2.6	29792	451.4
45	picnic_L5_FS	5	65	2	97	2	132860	2013
46	picnic_L5_UR	5	65	2	97	2	209510	3174.4
47	picnic_L5_full	5	65	2	97	2	126290	1913.5
48	picnic3_L5	5	65	2	97	2	61028	924.7
49	Falcon-512	1	897	28	1281	26.1	690	10.5
50	Dilithium2	2	1312	41	2528	51.6	2420	36.7
51	Dilithium2-AES	2	1312	41	2528	51.6	2420	36.7
52	Falcon-1024	5	1793	56	2305	47	1330	20.2
53	Dilithium3	3	1952	61	4000	81.6	3293	49.9
54	Dilithium3-AES	3	1952	61	4000	81.6	3293	49.9
55	Dilithium5	5	2592	81	4864	99.3	4595	69.6
56	Dilithium5-AES	5	2592	81	4864	99.3	4595	69.6
57	Rainbow-I-Circumzenithal	1	60192	1881	103648	2115.3	66	1
58	Rainbow-I-Compressed	1	60192	1881	64	1.3	66	1
59	Rainbow-I-Classic	1	161600	5050	103648	2115.3	66	1
60	Rainbow-III-Circumzenithal	3	264608	8269	626048	12776.5	164	2.5
61	Rainbow-III-Compressed	3	264608	8269	64	1.3	164	2.5

56	SPHINCS+-SHAKE256-128s-r	Hash	446377018	21068.4	3274888436	14123.3	6274742	85.5	0	0	5	5
57	Rainbow-I-Classic	MQ	492275357	23234.8	6220698	26.8	6573578	89.5	0	5	5	10
58	Rainbow-I-Compressed	MQ	542931029	25625.7	235804438	1016.9	7520590	102.4	0	0	3	3
59	Rainbow-I-Circumzenithal	MQ	544285874	25689.6	6271496	27	7651253	104.2	0	5	3	8
60	SPHINCS+-SHA256-256s-r	Hash	635319074	29986.3	1889802242	8150	15630510	212.9	0	0	3	3
61	SPHINCS+-SHAKE256-192s-r	Hash	661757883	31234.1	1701951015	7339.9	10728453	146.1	0	0	3	3
62	Rainbow-V-Classic	MQ	1707296542	80582.3	132498942	571.4	139224699	1896.2	0	3	0	3
63	Rainbow-III-Classic	MQ	2306828504	108879.4	59391493	256.1	61864256	842.6	0	3	3	6
64	Rainbow-III-Circumzenithal	MQ	3264803255	154094.6	59377096	256.1	70084535	954.5	0	3	3	6
65	Rainbow-III-Compressed	MQ	3292975486	155424.3	3593929070	15499.2	75246328	1024.8	0	0	0	0

1	Dilithium2-AES	28
2	Dilithium3-AES	28
3	Dilithium2	28
4	Dilithium5-AES	26
5	Dilithium3	24
6	picnic_L1_full	23
7	picnic_L1_FS	23
8	Dilithium5	21
9	picnic_L3_full	20
10	picnic_L5_full	20
11	picnic_L1_UR	20
12	picnic_L3_UR	20
13	picnic3_L3	18
14	picnic3_L5	18
15	picnic3_L1	16
16	picnic_L5_UR	16
17	picnic_L5_FS	16
18	picnic_L3_FS	16
19	Falcon-512	16
20	Falcon-1024	16
21	SPHINCS+-Haraka-12	13
22	SPHINCS+-Haraka-12	13
23	SPHINCS+-Haraka-19	13
24	SPHINCS+-Haraka-19	13
25	SPHINCS+-Haraka-25	11
26	SPHINCS+-Haraka-25	11
27	Rainbow-I-Classic	10
28	SPHINCS+-SHA256-12	9
29	SPHINCS+-SHA256-19	9

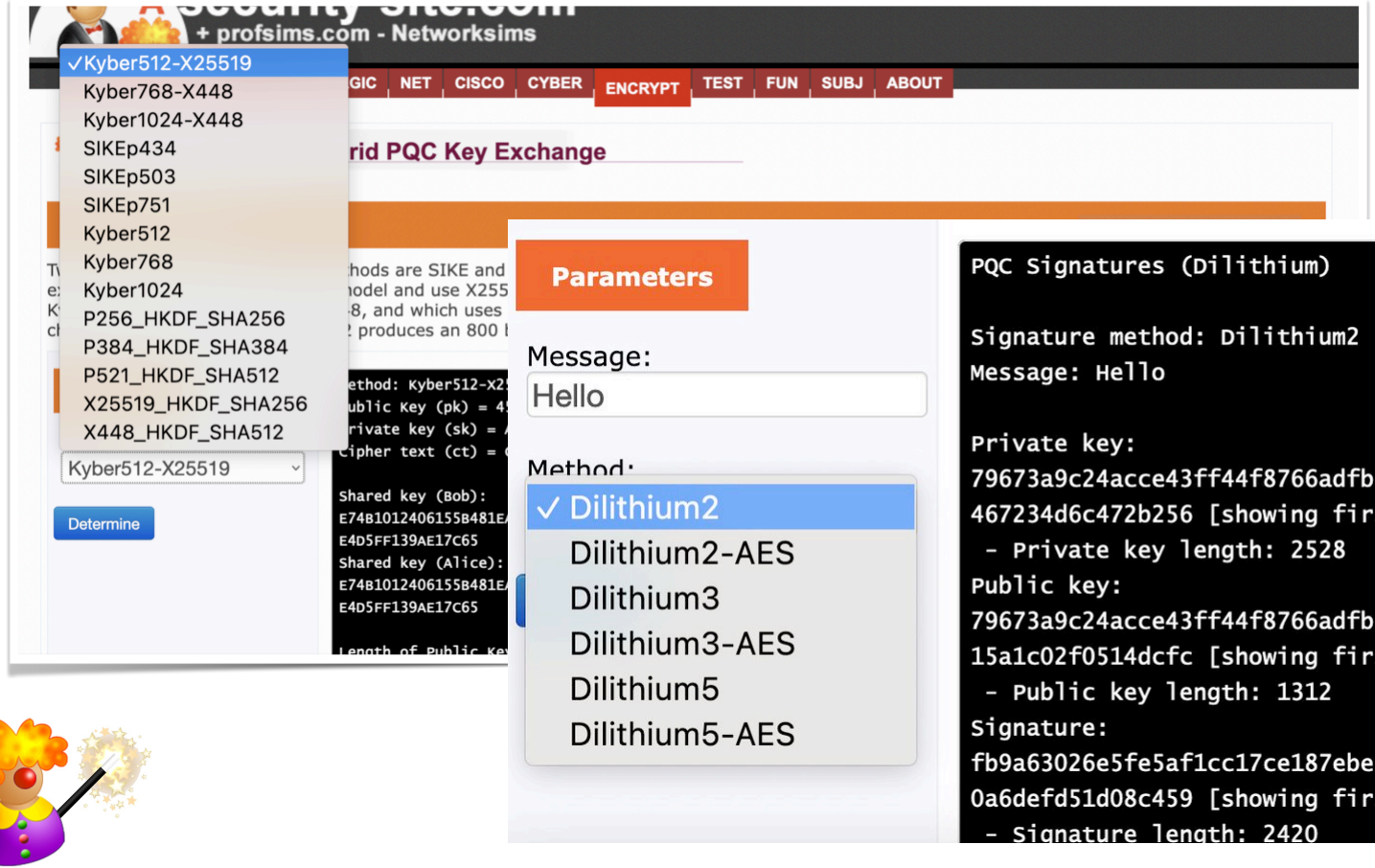
https://asecuritysite.com/pqc/

Hybrid Key Exchange/Digital Signatures



Hybrid

Hybrid Key Exchange/Digital Signatures



Hybrid

Key Exchange/Signatures

Type	Public key size (B)	Secret key size (B)	Ciphertext size (B)	
Kyber512	800	1,632	768	Learning with errors (Lattice)
Kyber738	1,184	2,400	1,088	Learning with errors (Lattice)
Kyber1024	1,568	3,168	1,568	Learning with errors (Lattice)
LightSABER	672	1,568	736	Learning with rounding (Lattice)
SABER	992	2,304	1,088	Learning with rounding (Lattice)
FireSABER	1,312	3,040	1,472	Learning with rounding (Lattice)
McEliece348864	261,120	6,452	128	Code based
McEliece460896	524,160	13,568	188	Code based
McEliece6688128	1,044,992	13,892	240	Code based
McEliece6960119	1,047,319	13,948	226	Code based
McEliece8192128	1,357,824	14,120	240	Code based
NTRUhps2048509	699	935	699	Lattice
NTRUhps2048677	930	1,234	930	Lattice
NTRUhps4096821	1,230	1,590	1,230	Lattice
SIKEp434	330	44	346	Isogeny
SIKEp503	378	56	402	Isogeny
SIKEp751	564	80	596	Isogeny
SIDH	564	48	596	Isogeny

Hybrid

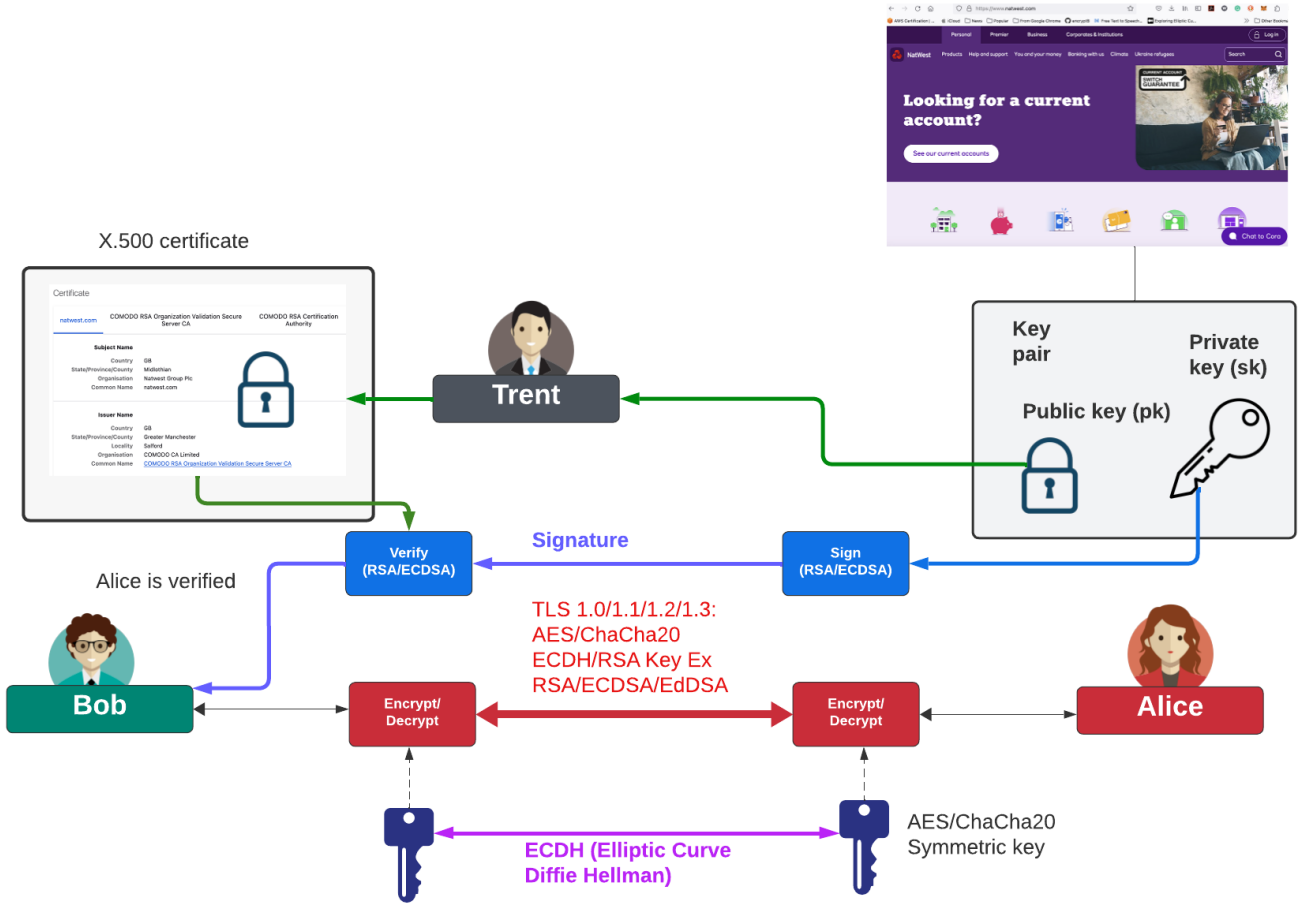
Key Exchange/Signatures

Type	Public key size (B)	Secret key size (B)	Ciphertext size (B)	
Kyber512	800	1,632	768	Learning with errors (Lattice)
Kyber738	1,184	2,400	1,088	Learning with errors (Lattice)
Kyber1024	1,568	3,168	1,568	Learning with errors (Lattice)
LightSABER	672	1,568	736	Learning with rounding (Lattice)
SABER	992	2,304	1,088	Learning with rounding (Lattice)
FireSABER	1,312	3,040	1,472	Learning with rounding (Lattice)
McEliece348864	261,120	6,452	128	Code based
McEliece460896	524,160	13,568	188	Code based
McEliece6688128	1,044,992	13,892	240	Code based
McEliece6960119	1,047,319			
McEliece8192128	1,357,824			
NTRUhps2048509	699			
NTRUhps2048677	930			
NTRUhps4096821	1,230			
SIKEp434	330			
SIKEp503	378			
SIKEp751	564			
SIDH	564			

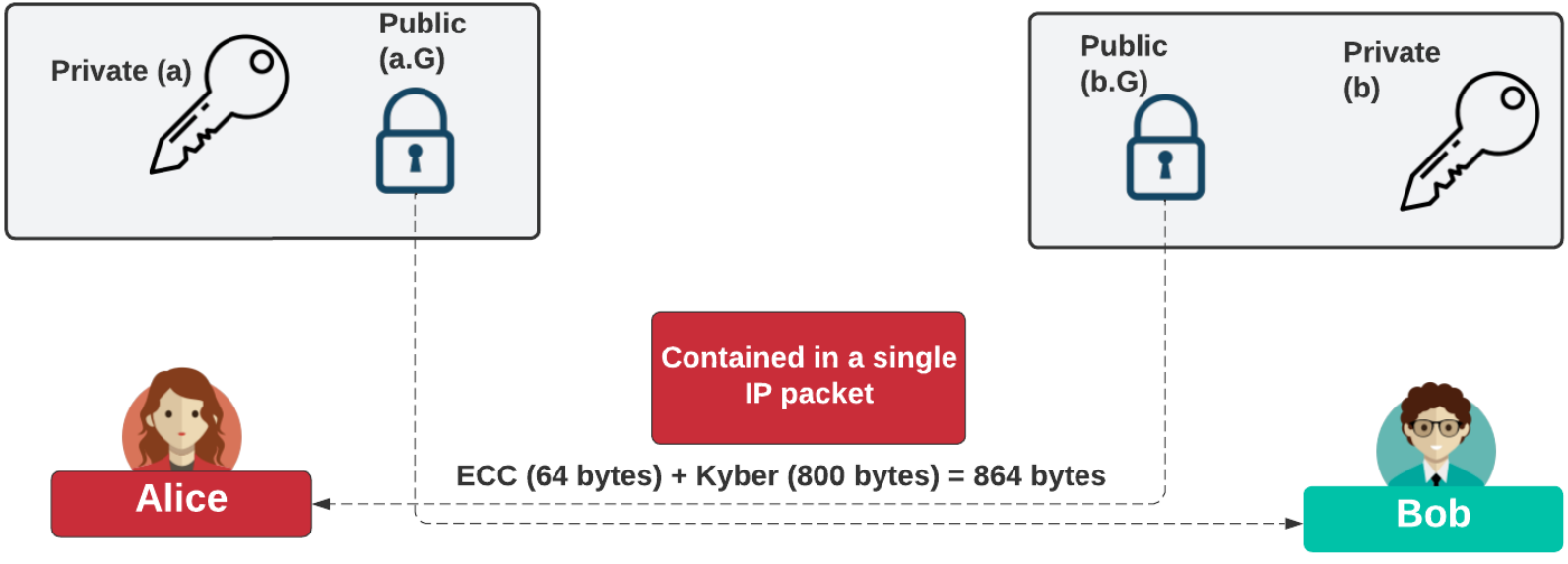
Method	Public key size	Private key size	Signature size	Security level
Crystals Dilithium 2 (Lattice)	1,312	2,528	2,420	1 (128-bit) Lattice
Crystals Dilithium 3	1,952	4,000	3,293	3 (192-bit) Lattice
Crystals Dilithium 5	2,592	4,864	4,595	5 (256-bit) Lattice
Falcon 512 (Lattice)	897	1,281	690	1 (128-bit) Lattice
Falcon 1024	1,793	2,305	1,330	5 (256-bit) Lattice
Rainbow Level Ia (Oil-and-Vineger)	161,600	103,648	66	1 (128-bit) Multivariate (UOV)
Rainbow Level IIIa	861,400	611,300	164	3 (192-bit) Multivariate (UOV)
Rainbow Level Vc	1,885,400	1,375,700	204	5 (256-bit) Multivariate (UOV)
Sphincs SHA256-128f Simple	32	64	17,088	1 (128-bit) Hash-based
Sphincs SHA256-192f Simple	48	96	35,664	3 (192-bit) Hash-based
Sphincs SHA256-256f Simple	64	128	49,856	5 (256-bit) Hash-based
Picnic 3 Full	49	73	71,179	3 (192-bit) Symmetric
GeMSS 128	352,188	16	33	1 (128-bit) Multivariate (HFEv-)
GeMSS 192	1,237,964	24	53	1 (128-bit) Multivariate (HFEv-)
RSA-2048	256	256	256	
ECC 256-bit	64	32	256	

Hybrid

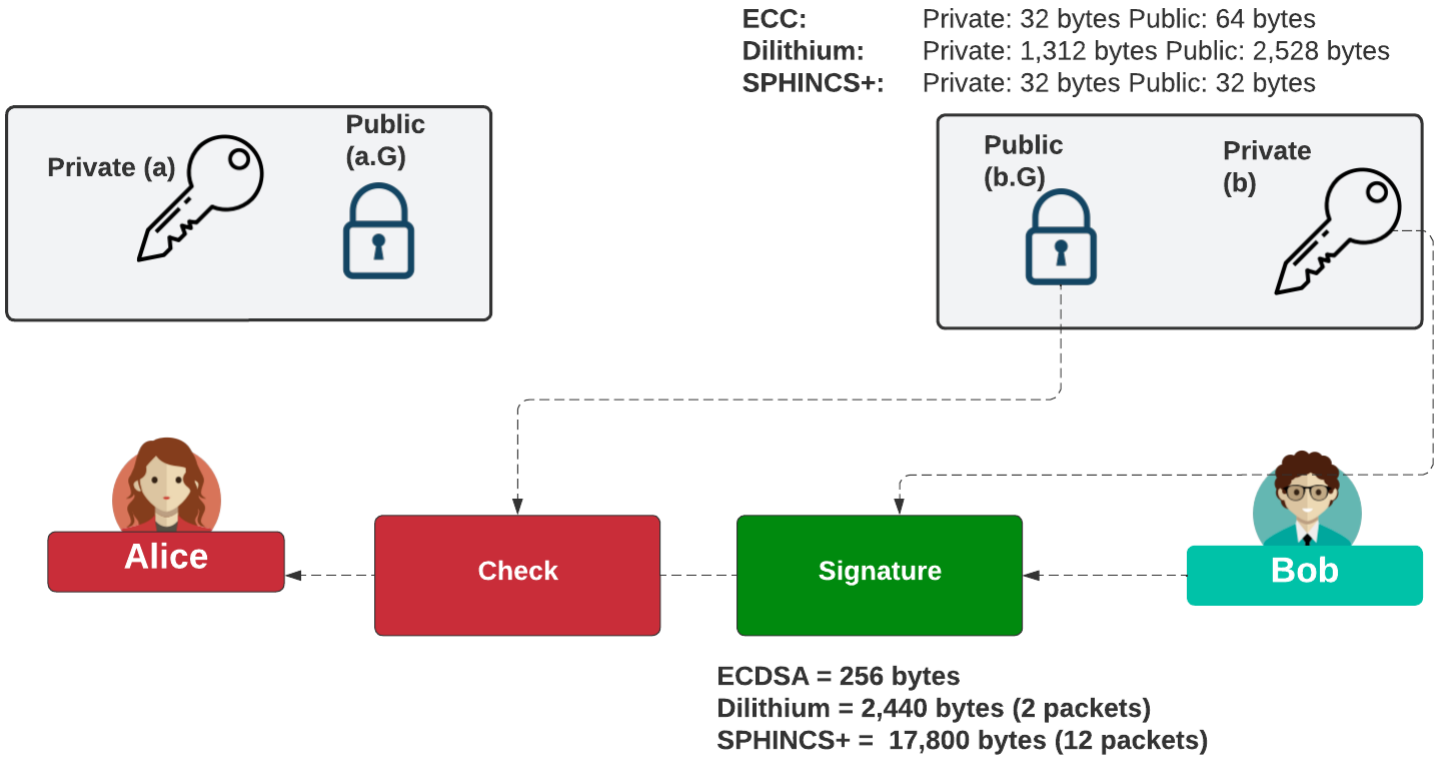
Key Exchange and Digital Signatures



Key Exchange



Digital Signatures



Methods

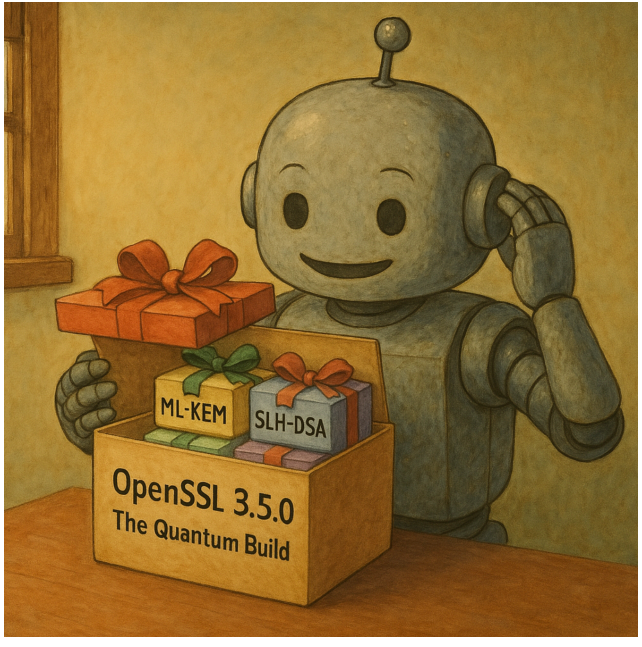
Key Exchange:

- ML-KEM (FIPS 203). https://asecuritysite.com/webcrypto/crypt_ml_kem
- HQC (FIPS 207?). https://asecuritysite.com/pqc/bc_hqc

Signatures:

- ML-DSA (FIPS 204) - Dlithium. https://asecuritysite.com/webcrypto/crypt_ml_dsa
- SLH-DSA (FIPS 205) - SPHINCS+. https://asecuritysite.com/csharp/bc_slh_dsa
- FN-DSA (FIPS 206) - FALCON. https://asecuritysite.com/webcrypto/crypt_falcon

OpenSSL 3.5



- OpenSSL 3.5: https://medium.com/asecuritysite-when-bob-met-alice/no-excuses-openssl-enters-the-quantum-age-ad29af287273
- OpenSSL 3.5 for Windows: https://github.com/billbuchanan/openssl

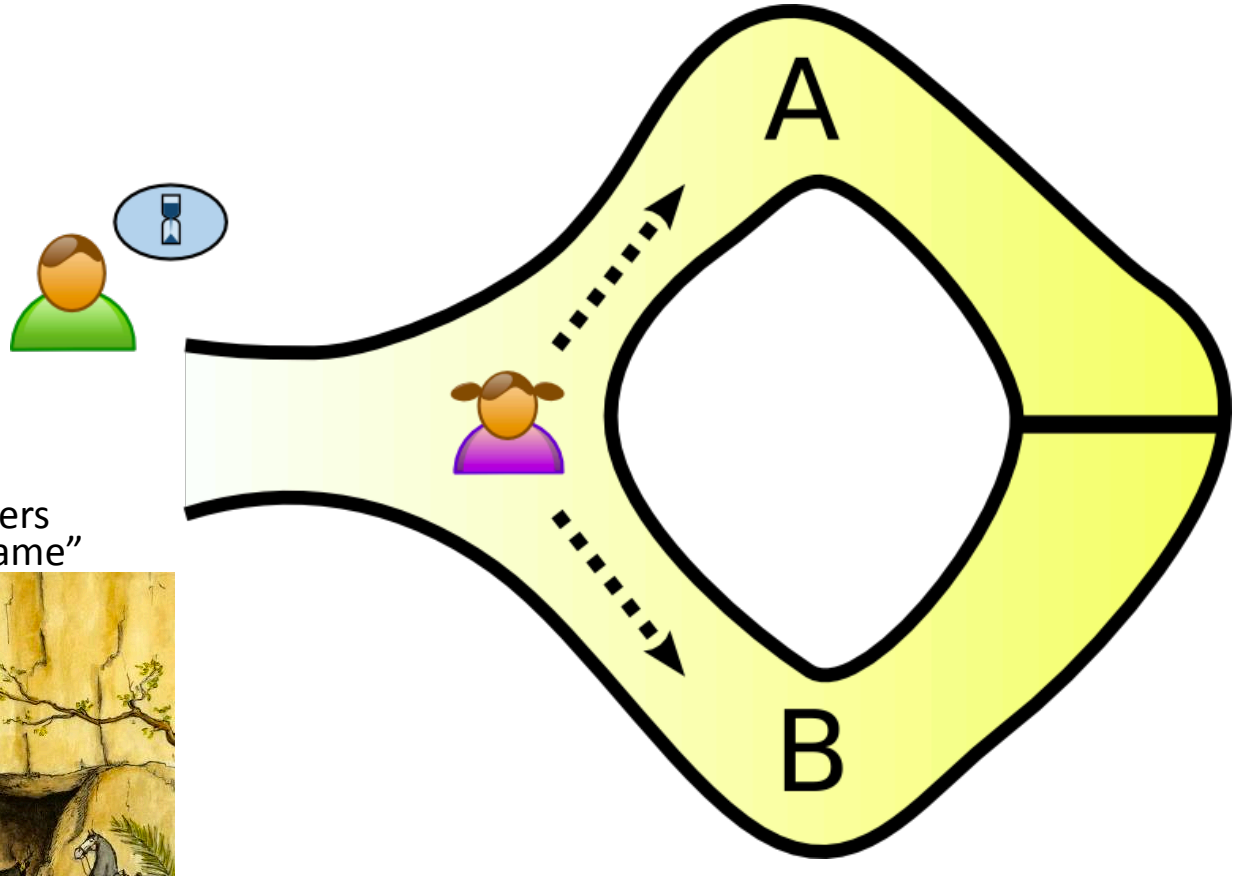
Next Generation Crypto

Light-weight Cryptography.
Post Quantum Cryptography
Zero-knowledge Proofs.
Homomorphic Encryption.
IPFS

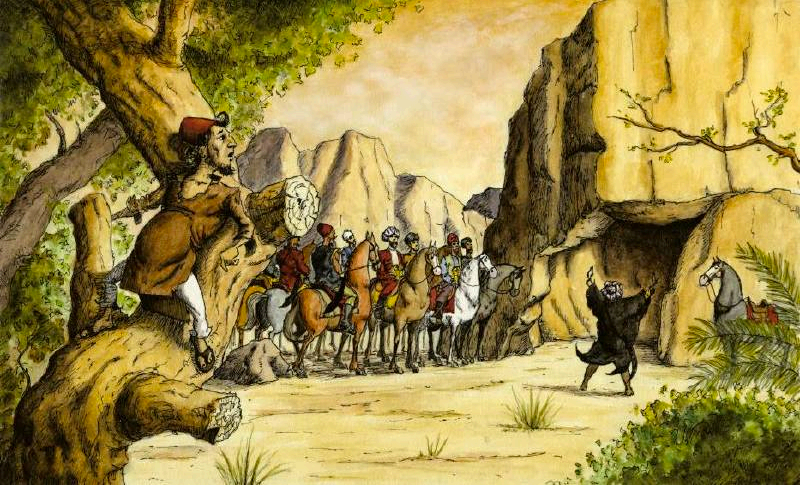
Prof Bill Buchanan OBE
http://asecuritysite.com/zero

Zero-knowledge Proof

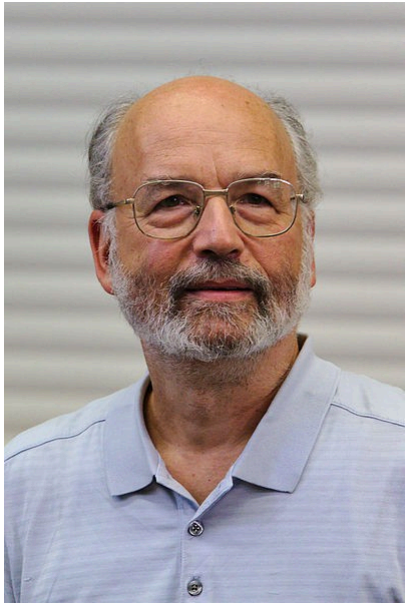
- Peggy is the prover.
- Victor is the verifier.



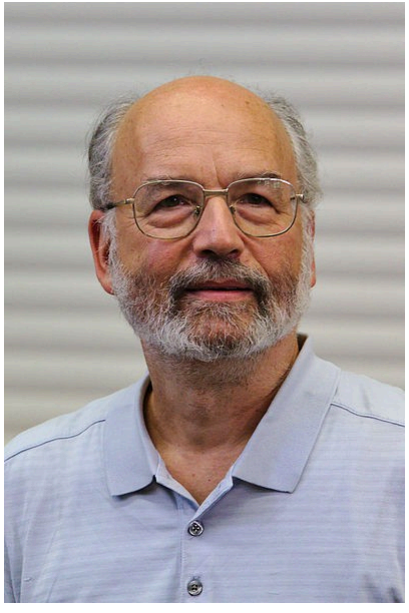
Ali Baba - poor woodcutter - discovers the secret of a thieves as “open sesame”



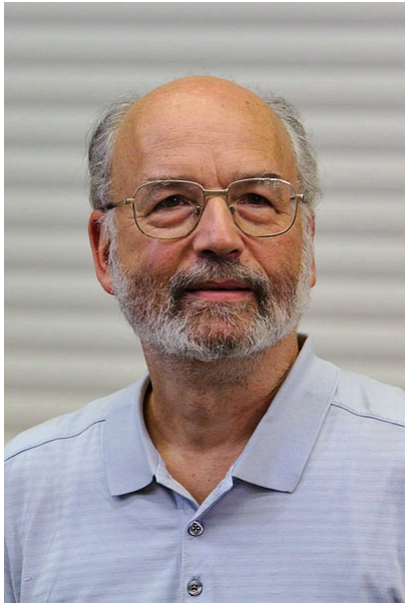
Zero Knowledge Proof: Fiat-Shamir

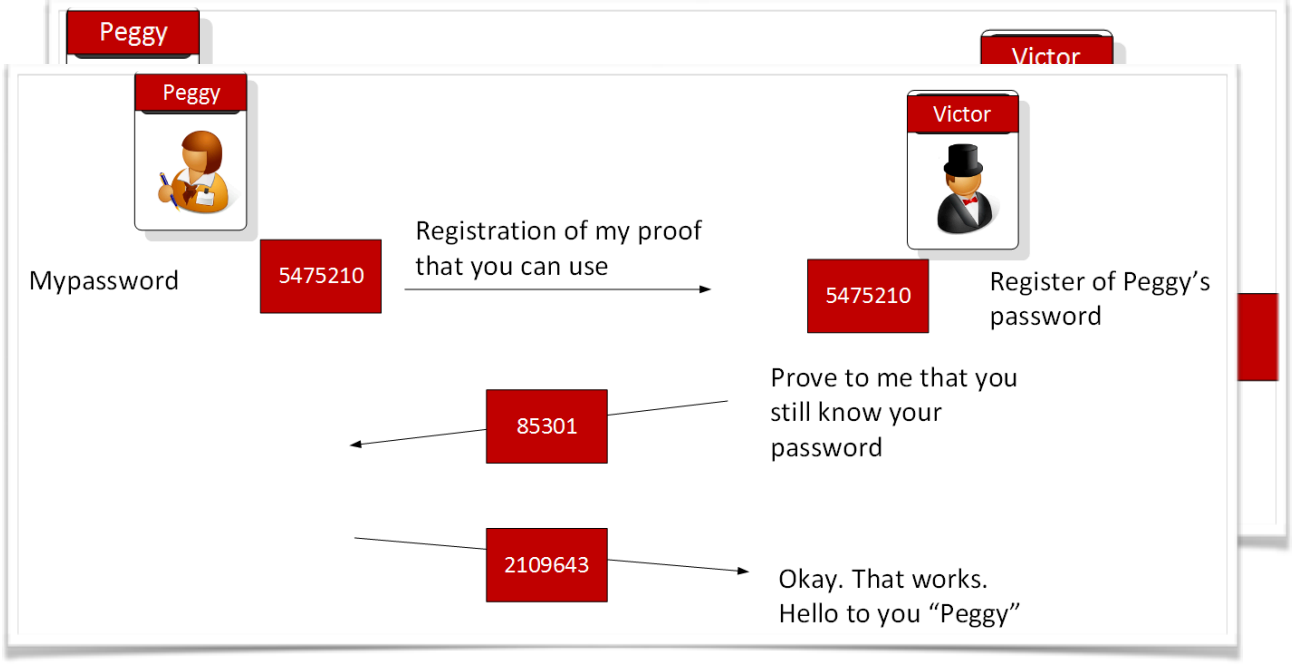


Zero Knowledge Proof: Fiat-Shamir

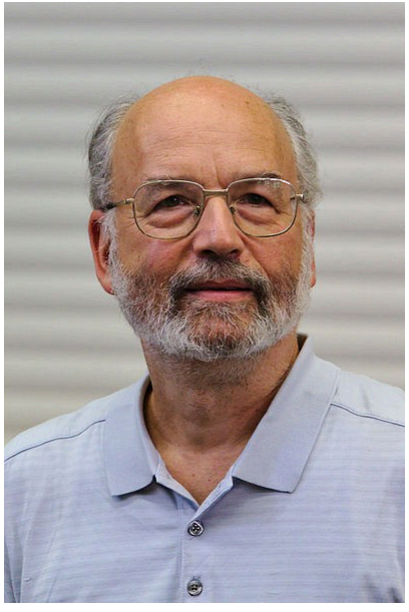


Zero Knowledge Proof: Fiat-Shamir





Zero Knowledge Proof: Fiat-Shamir



Peggy

Victor

pass= Mypassword
x = Hash(pass)
$y = g^x$

$y = g^x$

$y = g^x$

Peggy select *v*

$t = g^v$

Victor generates a random value (c)

c

$r = v - c\,x$

r

Victor computes:
$g^r y^c$

If same as t, Peggy is verified

Bob proves and Alice verifies - Non-interactive random oracle access

Bob is the prover and knows the value of *x*

Bob and Alice agree on G and p

Alice is the verifier

Shared Secret: $y = G^x$

Prove to me you still know *x*!

Random value: *v*

Commitment: $t=g^v$

Challenge: c =Hash(g,y,t)

Response: r =v – c x (mod p)

$$g^r y^c = g^{v-cx} y^c = g^{v-cx} (g^x)^c = g^{v-cx+cv} = t$$

(r,c)

Check $t' = g^r y^c$

Recheck c = H(g,y,t')

Bob proves anc

Peggy and Victor agree on a base point: G

Bob is the prover an knows the of *x*

Secret: x
Public value: =x.G

pk=x.G

Random

Peggy selects *v*

Commiti

c=H(G,vG,xG,userID)

Challeng

r = v - c.x (mod n)

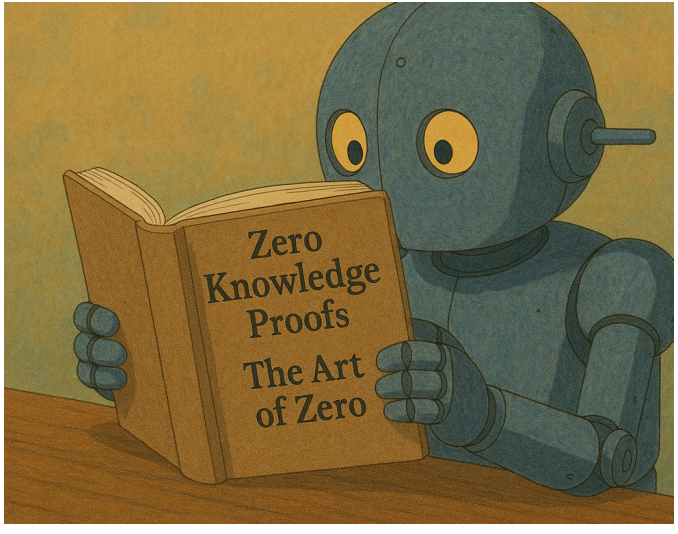
Respons

r, vG

Victor checks
vG = rG + c (xG)

Proof:
vG = rG + c (xG) = (v-cx)G + cx G = vG

Zero Knowledge Proofs



- Zero Knowledge Proofs: https://asecuritysite.com/zero

Next Generation Crypto

Light-weight Cryptography.
Post Quantum Cryptography
Zero-knowledge Proofs.
> Homomorphic Encryption.
IPFS

Prof Bill Buchanan OBE

http://asecuritysite.com/homomorphic

Polynomials

$$Enc_k(A \circ B) = Enc_k(A) \circ Enc_k(B)$$

With lattices, we use polynomials to represent our multi-dimensional spaces. In general, a polynomial can be represented by a number of coefficients (a_n ... a_0) and polynomial powers:

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + ...a_1 x + a_0 \tag{2.33}$$

Within a lattice, we can have a point at (9, 5, 16), and then represent it with a quadratic equation of:

$$16x^2 + 5x + 9 \tag{2.34}$$

$$f = (16x^2 + 5x + 9)(2x^2 + x + 7) = 32x^4 + 26x^3 + 135x^2 + 44x + 63 \tag{2.35}$$

$$\begin{aligned} A \times B &= (10x^2 + 6x + 2) \times (12x^2 + 3x + 2) \\ &= 120x^4 + 30x^3 + 20x^2 + 72x^3 + 180x^2 + 12x + 24x^2 + 6x + 4 \\ &= 120x^4 + 102x^3 + 62x^2 + 18x + 4 \end{aligned}$$

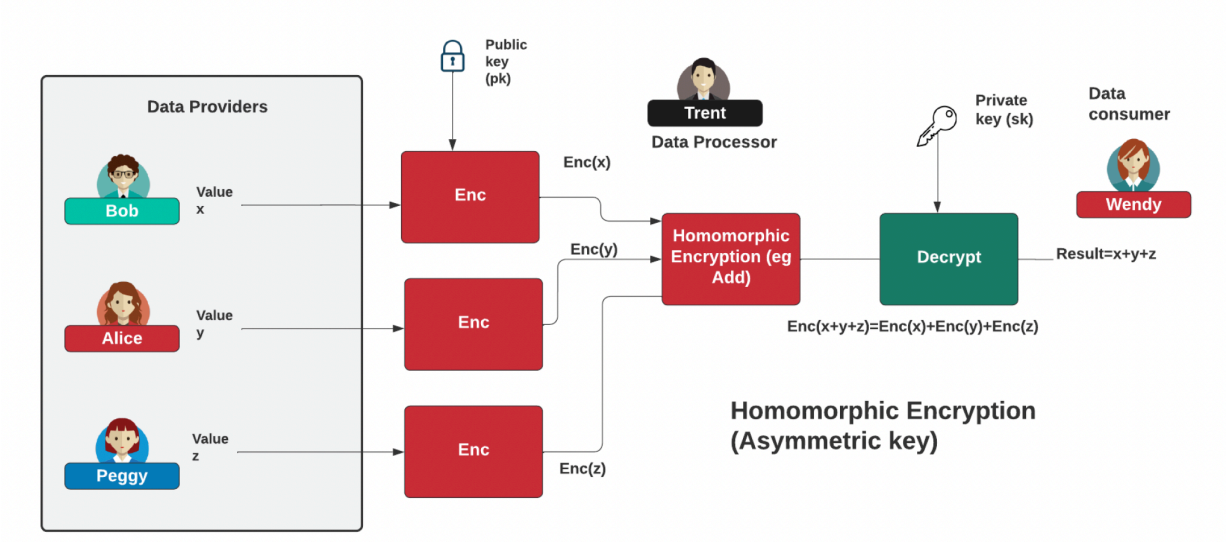
And now, we can apply a $\pmod{13}$ operation to each of the coefficients:

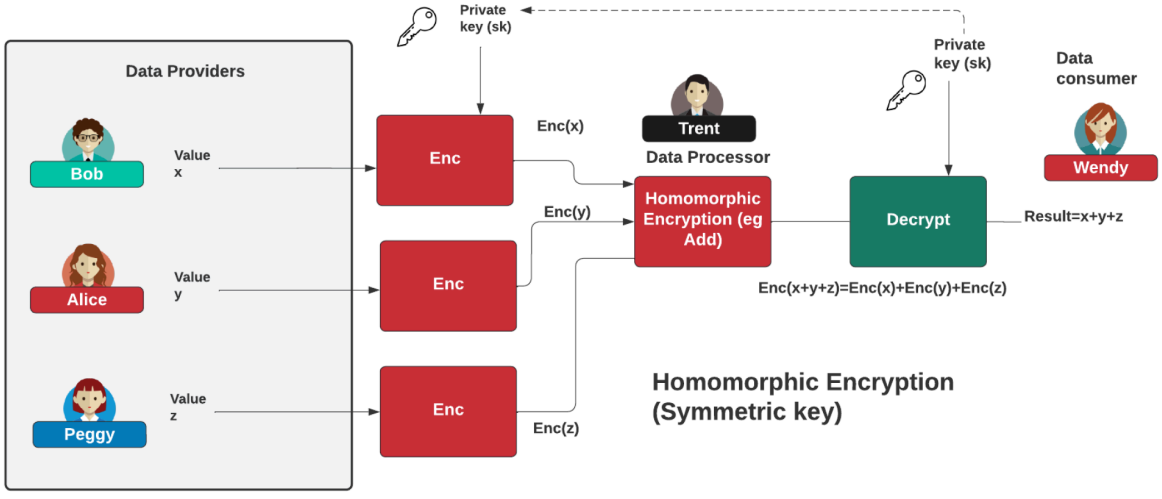
$$A \times B = 120x^4 + 102x^3 + 62x^2 + 18x + 4 = 3x^4 + 11x^3 + 10x^2 + 5x + 4 \pmod{13}$$

FHE

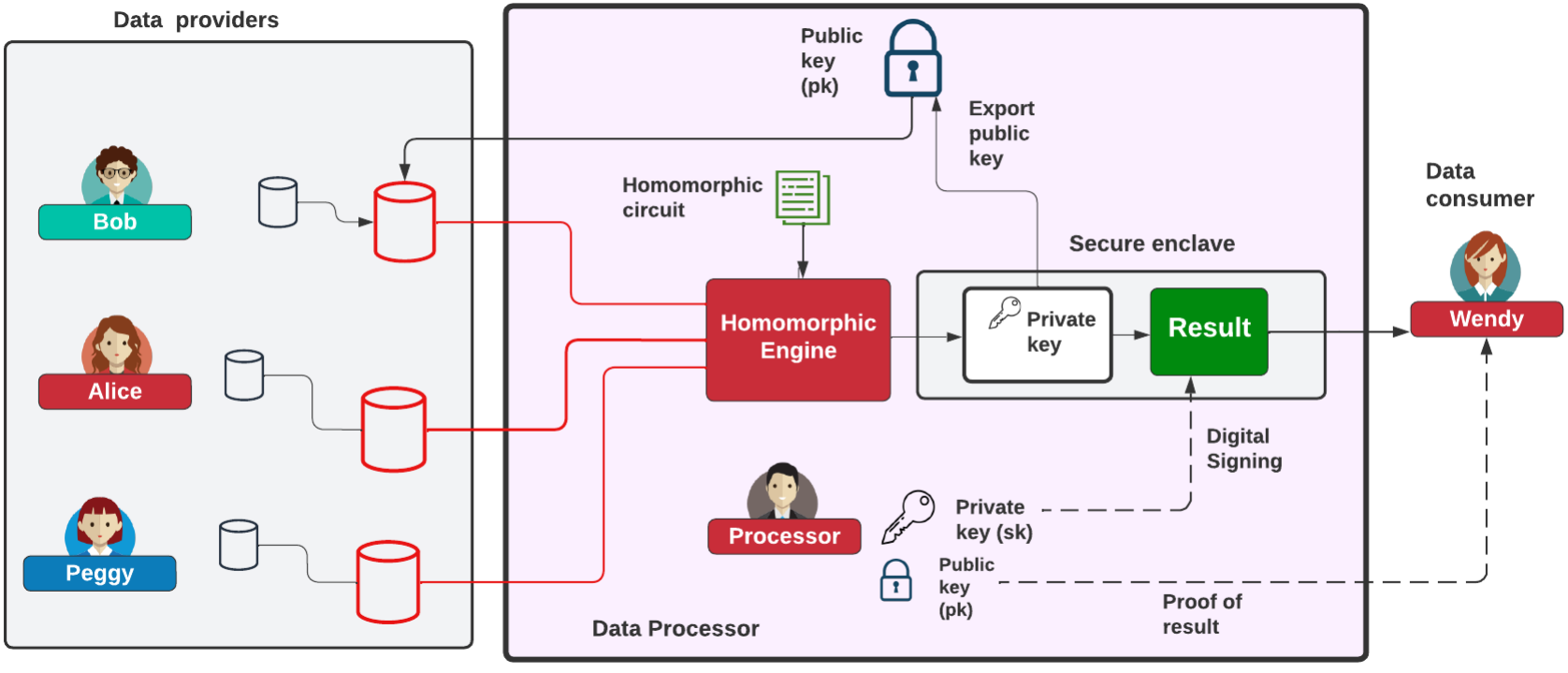
- 1st generation: Gentry's method uses integers and lattices [50] including the DGHV method.
- 2nd generation. Brakerski, Gentry and Vaikuntanathan's (BGV) and Brakerski/Fan-Vercauteren (BFV) use a Ring Learning With Errors approach [51]. The methods are similar to each other, and with only small difference between them.
- 3rd generation: These include DM (also known as FHEW) and CGGI (also known as TFHE) and support the integration of Boolean circuits for small integers.
- 4th generation: CKKS (Cheon, Kim, Kim, Song) and which uses floating-point numbers [52].

FHE

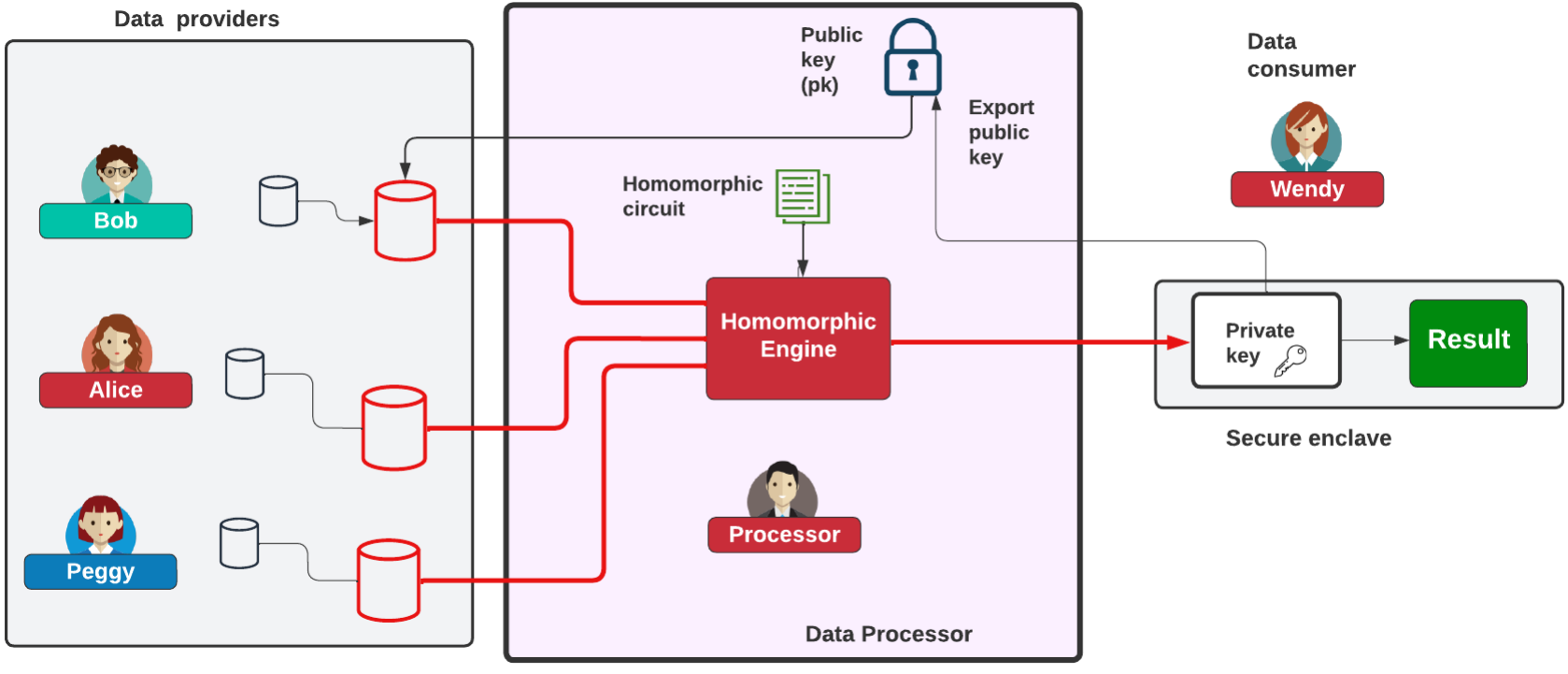

Data Providers
Bob
Value x
Alice
Value y
Peggy
Value z
Public key (pk)
Enc
Enc
Enc
Enc(x)
Enc(y)
Enc(z)
Trent
Data Processor
Homomorphic Encryption (eg Add)
Private key (sk)
Decrypt
Result=x+y+z
Data consumer
Wendy
Enc(x+y+z)=Enc(x)+Enc(y)+Enc(z)
Homomorphic Encryption (Asymmetric key)


Data Providers
Bob
Value x
Alice
Value y
Peggy
Value z
Private key (sk)
Enc
Enc
Enc
Enc(x)
Enc(y)
Enc(z)
Trent
Data Processor
Homomorphic Encryption (eg Add)
Private key (sk)
Decrypt
Result=x+y+z
Data consumer
Wendy
Enc(x+y+z)=Enc(x)+Enc(y)+Enc(z)
Homomorphic Encryption (Symmetric key)

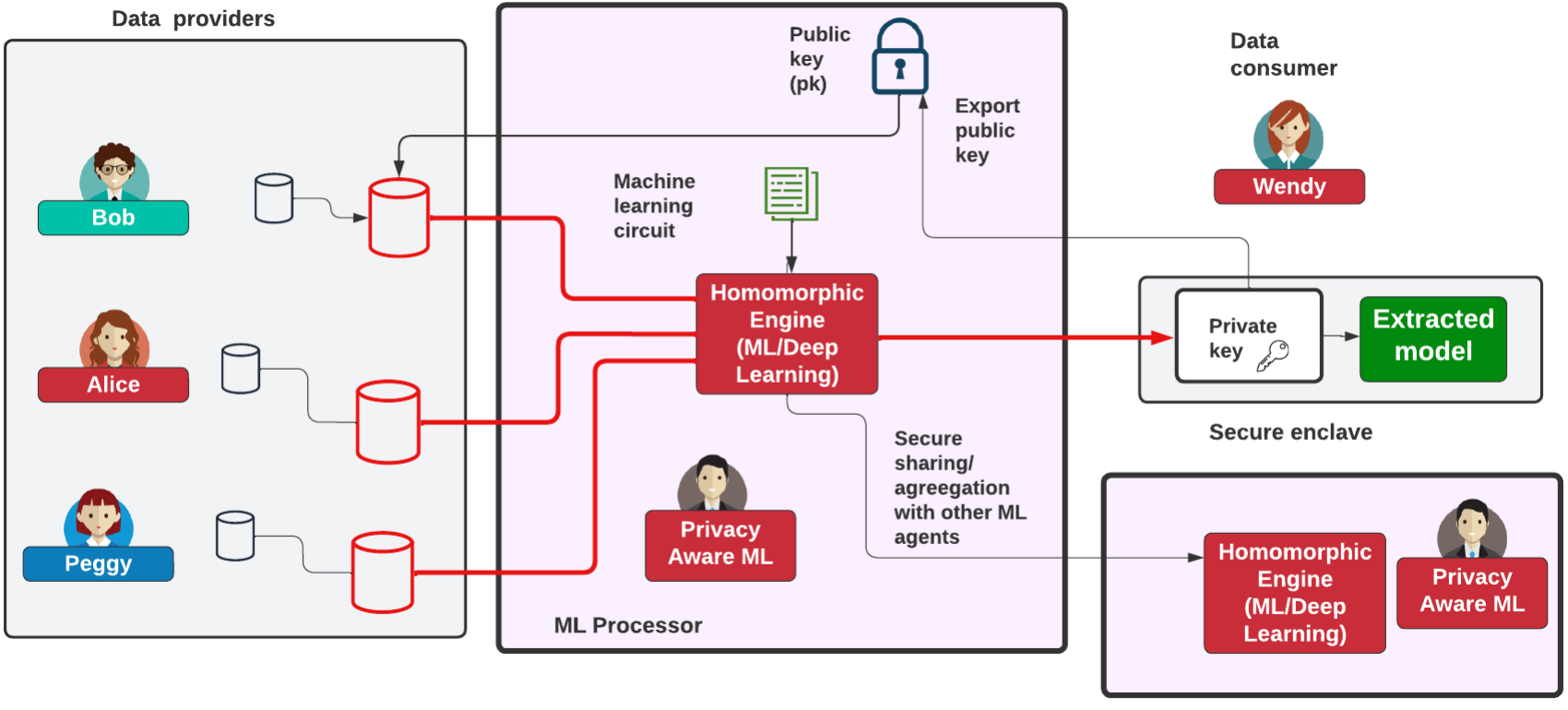
Homomorphic Processing



Homomorphic Processing



Homomorphic ML



Libraries

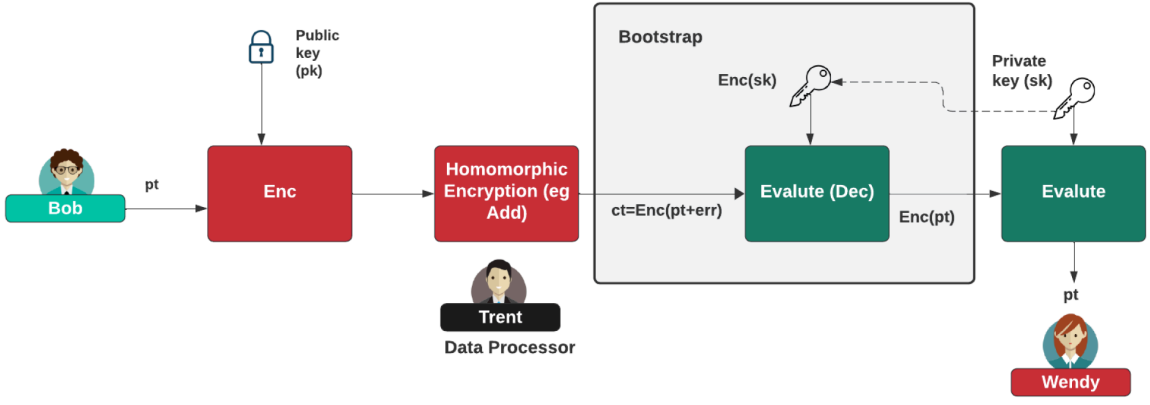
Product	Creator	Language	License	Summary
SEAL [177]	Microsoft	C++	MIT	Widely-used FHE library that implements BFV for modular arithmetic and CKKS for approximate arithmetic.
HElib [116]	IBM	C++	Apache-2.0	Widely-used FHE library that implements BGV for modular arithmetic and CKKS for approximate arithmetic.
TFHE [59]	Gama et al.	C++	Apache-2.0	Implements an optimized ring variant of the GSW scheme.
HEAAN [115]	CryptoLab, Inc.	C++	CC-BY-NC-3.0	Implements the CKKS approximate number arithmetic scheme.
PALISADE [165]	New Jersey Institute of Technology	C++	BSD-2-Clause	Lattice cryptography library that supports multiple protocols for FHE, including BGV, BFV, and StSt.
$\Lambda \circ \lambda$ [69]	E. Crockett and C. Peikert	Haskell	GPL-3.0-only	Pronounced "LOL." Implements a BGV-type FHE scheme.
Cingulata [45]	CEA LIST	C++	CECILL-1.0	Compiler and RTE for C++ FHE programs. Implements BFV and supports TFHE.
FV-NFLlib [89]	CryptoExperts	C++	GPL-3.0-only	Implements FV scheme. Built on the NFLLib lattice cryptography library. Last updated 2016.
Lattigo [138]	Laboratory for Data Security	Go	Apache 2.0	Implements BFV and HEAAN in Go.

OpenFHE

OpenFHE

- Brakerski/Fan-Vercauteren (**BFV**) scheme for integer arithmetic
- Brakerski-Gentry-Vaikuntanathan (**BGV**) scheme for integer arithmetic
- Cheon-Kim-Kim-Song (**CKKS**) scheme for real-number arithmetic (includes approximate bootstrapping)
- Ducas-Micciancio (**DM**) and Chillotti-Gama-Georgieva-Izabachene (**CGGI**) schemes for Boolean circuit evaluation.

Bootstrapping



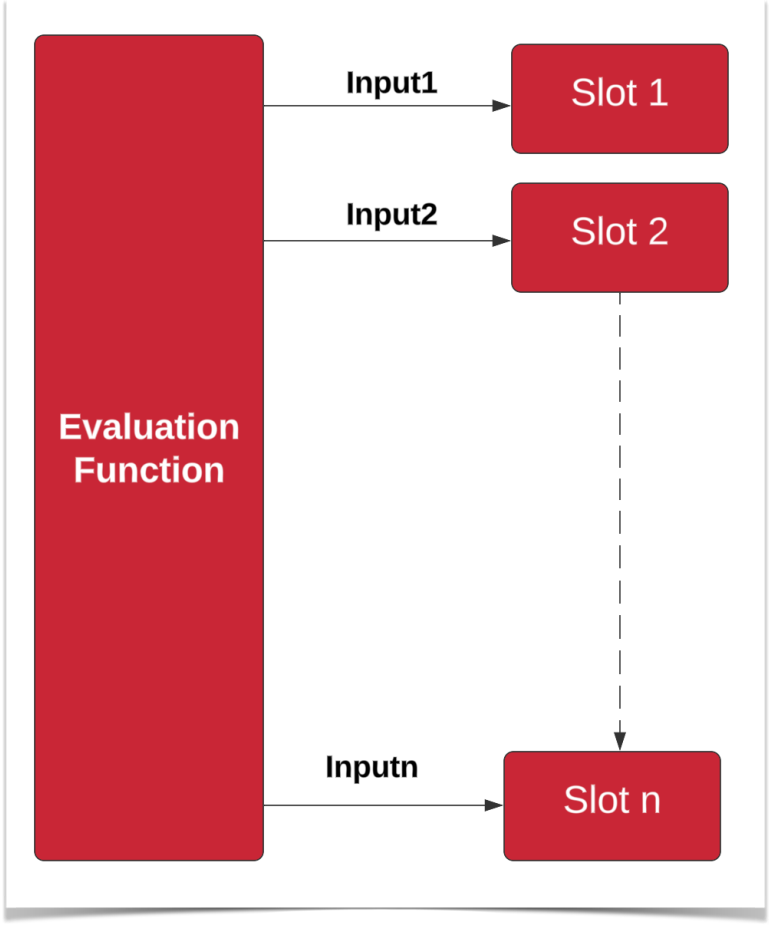
The main bootstrapping methods are CKKS [52], DM [57]/CGGI, and BGV/BFV. Overall, CKKS is generally the fastest bootstrapping method, while DM/CGGI is efficient with the evaluation of arbitrary functions. These functions approximate maths functions as polynomials (such as with Chebyshev approximation). BGV/BFV provides reasonable performance and is generally faster than DM/CGGI but slower than CKKS.

Bootstrapping and Slots

Each ciphertext can have an associated "level" and a value of "noise".

We have various levels. One multiplication consumes a level, and adds noise.

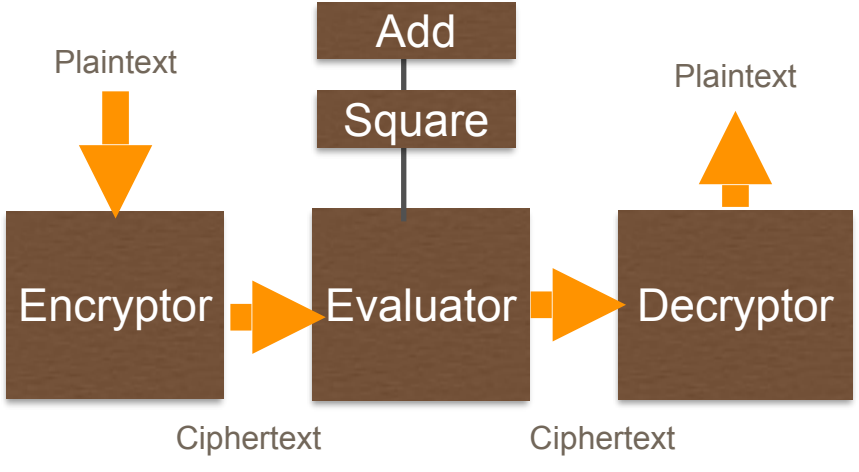
When we reach zero or the noise level is too high, we need to bootstrap - raises the level and reduce noise.



SEAL BFV/BGV

PolyModulusDegree: Defines number of coefficients in plaintext polynomials and the size of ciphertext elements. Must be a power of 2 (such as 1024, 2048, 4096, 8192, 16384, or 32768). Affects performance)

PlainModulus: Largest coefficient that plaintext polynomials can represent. Affects performance.

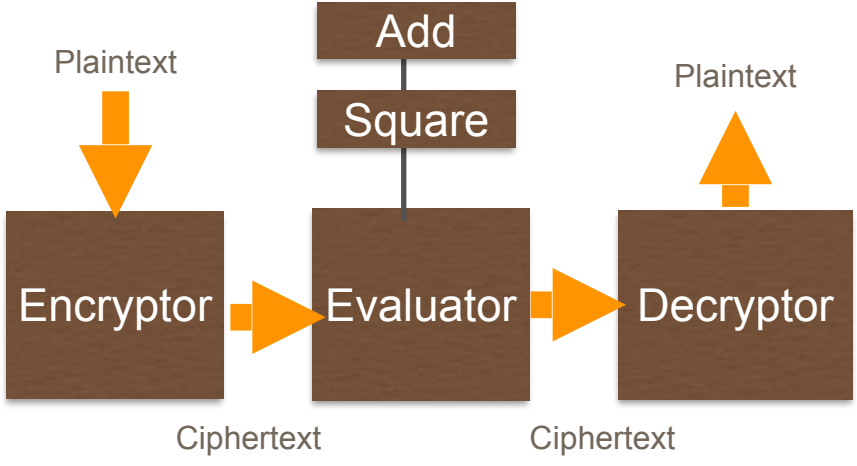



```
EncryptionParameters parms = new EncryptionParameters(SchemeType.BFV);

if (type=="BFV")  parms = new EncryptionParameters(SchemeType.BFV);
else if  (type=="BGV")  parms = new EncryptionParameters(SchemeType.BGV);


Console.WriteLine("Example: {0}\n",type);

ulong polyModulusDegree = mod;
parms.PolyModulusDegree = polyModulusDegree;


parms.CoeffModulus = CoeffModulus.BFVDefault(polyModulusDegree);


parms.PlainModulus = new Modulus(2024);


using SEALContext context = new SEALContext(parms);

using KeyGenerator keygen = new KeyGenerator(context);
using SecretKey secretKey = keygen.SecretKey;
keygen.CreatePublicKey(out PublicKey publicKey);

using Encryptor encryptor = new Encryptor(context, publicKey);
using Evaluator evaluator = new Evaluator(context);
using Decryptor decryptor = new Decryptor(context, secretKey);


using Ciphertext xEncrypted = new Ciphertext();


using Plaintext xPlain = new Plaintext(ULongToString(x));
```

SEAL CKKS

PolyModulusDegree: Defines number of coefficients in plaintext polynomials and the size of ciphertext elements. Must be a power of 2 (such as 1024, 2048, 4096, 8192, 16384, or 32768). Affects performance)

```
using EncryptionParameters parms = new EncryptionParameters(SchemeType.CKKS);


ulong polyModulusDegree = 8192;
parms.PolyModulusDegree = polyModulusDegree;
parms.CoeffModulus = CoeffModulus.Create(
    polyModulusDegree, new int[]{ 60, 40, 40, 60 });


double scale = Math.Pow(2.0, 40);

using SEALContext context = new SEALContext(parms);

Console.WriteLine();

using KeyGenerator keygen = new KeyGenerator(context);
using SecretKey secretKey = keygen.SecretKey;
keygen.CreatePublicKey(out PublicKey publicKey);
keygen.CreateRelinKeys(out RelinKeys relinKeys);
using Encryptor encryptor = new Encryptor(context, publicKey);
using Evaluator evaluator = new Evaluator(context);
using Decryptor decryptor = new Decryptor(context, secretKey);



using CKKSEncoder encoder = new CKKSEncoder(context);
ulong slotCount = encoder.SlotCount;
```



CryptoContext and Parameters (BFV)

Plaintext Modulus

Multiplicative Depth

PKE Scheme Features

Generate key pair (sk, pk)

```
CCParams<CryptoContextBFVRNS> parameters;
parameters.SetPlaintextModulus(mod);
parameters.SetMultiplicativeDepth(2);

CryptoContext<DCRTPoly> cryptoContext = GenCryptoContext(parameters);

cryptoContext->Enable(PKE);
cryptoContext->Enable(KEYSWITCH);
cryptoContext->Enable(LEVELEDSHE);


KeyPair<DCRTPoly> keyPair;

// Generate a public/private key pair
keyPair = cryptoContext->KeyGen();

std::cout << "The key pair has been generated." << std::endl;

auto str = Serial::SerializeToString(  cryptoContext);

cout << "Crypto Context (First 2,000 characters):\n" << str.substr(0,2000) << endl;
```

https://asecuritysite.com/openfhe/openfhe_00cpp

CryptoContext and Parameters (CKKS)

Multiplication depth

Scale Mod Size

```
uint32_t multDepth = 1;
uint32_t scaleModSize = 50;

if (argc>1) {
    std::istringstream iss(argv[1]);
    iss >>scaleModSize;
}

CCParams<CryptoContextCKKSRNS> parameters;
parameters.SetMultiplicativeDepth(multDepth);
parameters.SetScalingModSize(scaleModSize);

CryptoContext<DCRTPoly> cryptoContext = GenCryptoContext(parameters);

cryptoContext->Enable(PKE);
cryptoContext->Enable(KEYSWITCH);
cryptoContext->Enable(LEVELEDSHE);

KeyPair<DCRTPoly> keyPair;

// Generate a public/private key pair
keyPair = cryptoContext->KeyGen();

std::cout << "The key pair has been generated." << std::endl;

auto str = Serial::SerializeToString(  keyPair.publicKey);
```

https://asecuritysite.com/openfhe/openfhe_00cpp_ckks

BFV - Adding/Multiplying Two Numbers

```
// Multiply ciphertext
auto ciphertextMult     = cryptoContext->(ciphertext1, ciphertext2);

// Decrypt result
Plaintext plaintextMultRes;
cryptoContext->Decrypt(keyPair.secretKey,

std::cout << "Method: : " << type << std:
std::cout << "Modulus: : " << mod<< std::

std::cout << "\nx: " << xplaintext << std
```

Encrypt
EvalAdd
EvalAddInPlace
EvalAddMany
EvalAddManyInPlace
EvalAddMutable
EvalAddMutableInPlace
EvalAtIndex
EvalAtIndexKeyGen
EvalAutomorphism
EvalAutomorphismKeyGen
EvalBootstrap

S 2 OUTPUT TERMINAL DEBUG CONSOLE

2024, 12:18:22] Unable to resolve configurat
ng" instead.

```
keyPair = cryptoContext->KeyGen();


std::vector<int64_t>xval = {1};
xval[0]=x;
Plaintext xplaintext                = cryptoContext->MakePackedPlaintext(xval);

std::vector<int64_t> yval = {1};
yval[0]=y;
Plaintext yplaintext                = cryptoContext->MakePackedPlaintext(yval);

// Encrypt values
auto ciphertext1 = cryptoContext->Encrypt(keyPair.publicKey, xplaintext);
auto ciphertext2 = cryptoContext->Encrypt(keyPair.publicKey, yplaintext);

// Add ciphertext
auto ciphertextMult     = cryptoContext->EvalAdd(ciphertext1, ciphertext2);

// Decrypt result
Plaintext plaintextAddRes;
cryptoContext->Decrypt(keyPair.secretKey, ciphertextMult, &plaintextAddRes);
```

https://asecuritysite.com/openfhe/openfhe_02cpp

CCKS - Adding/Multiplying Two Numb

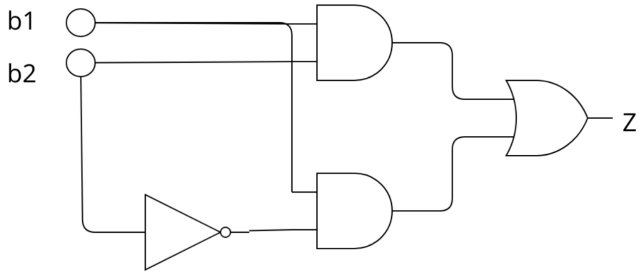
```
std::vector<double> x1 = {x};
std::vector<double> y1 = {y};

// Encoding as plain
Plaintext ptxt1 = cc
Plaintext ptxt2 = cc

std::cout << "Input
std::cout << "Input

// Encrypt the encod
auto c1 = cc->Encryp
auto c2 = cc->Encryp

//  Addition
auto cAdd = cc->Eval(c1, c2);
//  Subtraction
auto cSub = cc->EvalSub(c1, c2);
// Multiplication
```

```
EvalAdd          inline lbcrypto::Ciphertext<lbcrypto::DCRTPo…
EvalAddInPlace
EvalAddMany
EvalAddManyInPlace
EvalAddMutable
EvalAddMutableInPlace
EvalAtIndex
EvalAtIndexKeyGen
EvalAutomorphism
EvalAutomorphismKeyGen
EvalBootstrap
EvalBootstrapKeyGen
```

```
auto keys = cc->KeyGen();

cc->EvalMultKeyGen(keys.secretKey);

std::vector<double> x1 = {x};
std::vector<double> y1 = {y};

// Encoding as plaintexts
Plaintext ptxt1 = cc->MakeCKKSPackedPlaintext(x1);
Plaintext ptxt2 = cc->MakeCKKSPackedPlaintext(y1);

std::cout << "Input x1: " << ptxt1 << std::endl;
std::cout << "Input y1: " << ptxt2 << std::endl;

// Encrypt the encoded vectors
auto c1 = cc->Encrypt(keys.publicKey, ptxt1);
auto c2 = cc->Encrypt(keys.publicKey, ptxt2);

//  Addition
auto cAdd = cc->EvalAdd(c1, c2);
//  Subtraction
auto cSub = cc->EvalSub(c1, c2);
// Multiplication
auto cMul = cc->EvalMult(c1, c2);

Plaintext result;
std::cout.precision(8);
std::cout << std::endl << "Results: " << std::endl;
cc->Decrypt(keys.secretKey, cAdd, &result);
result->SetLength(batchSize);
std::cout << "x+y=" << result << std::endl;


cc->Decrypt(keys.secretKey, cSub, &result);
result->SetLength(batchSize);
std::cout << "x-y=" <<  result << std::endl;
```

https://asecuritysite.com/openfhe/openfhe_05cpp

MD/FHEW



$$(b_1 . b_2) + (b_1 . \bar{b_2})$$

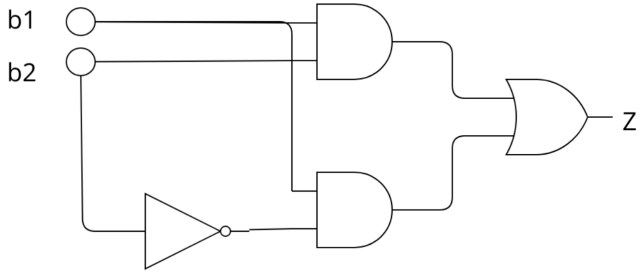
b1	b2	(b1.b2)	(b1.NOT(b2)	Z
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	0	1

```
auto sk = cc.KeyGen();

std::cout << "Creating bootstrapping keys..." << std::endl;

cc.BTKeyGen(sk);

std::cout << "Completed key generation." << std::endl;

auto bit1 = cc.Encrypt(sk, b1);
auto bit2 = cc.Encrypt(sk, b2);

cout << bit1 << endl;

auto ctAND1 = cc.EvalBinGate(AND, bit1, bit2);
auto bit2Not = cc.EvalNOT(bit2);
auto ctAND2 = cc.EvalBinGate(AND, bit2Not, bit1);
auto ctResult = cc.EvalBinGate(OR, ctAND1, ctAND2);

LWEPlaintext result;

cc.Decrypt(sk, ctResult, &result);

printf("b1=%d\n",b1);
printf("b2=%d\n",b2);
printf("(b1 AND b2) OR ( b1 AND NOT(b2))\n");
printf("(%d AND %d) OR ( %d AND NOT(%d))=%d\n",b1,b2,b1,b2,result);
```

https://asecuritysite.com/openfhe/openfhe_09cpp

https://asecuritysite.com/openfhe/openfhe_09cpp_pke

MD/FHEW



$$(b_1 . b_2) + (b_1 . \bar{b_2})$$

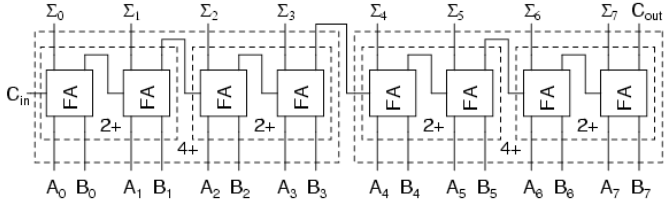
b1	b2	(b1.b2)	(b1.NOT(b2)	Z
0	0	0	0	0
0	1	0	0	0
1	0	0	1	1
1	1	1	0	1

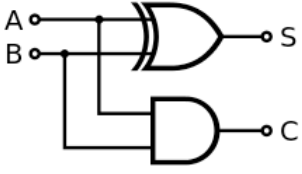
```
auto sk = cc.KeyGen();

std::cout << "Generating the bootstrapping keys... public keys" << std::endl;

// Generate the bootstrapping keys (refresh, switching and public keys)
cc.BTKeyGen(sk, PUB_ENCRYPT);

auto bit1 = cc.Encrypt(cc.GetPublicKey(), b1);
auto bit2 = cc.Encrypt(cc.GetPublicKey(), b2);

cout << bit1 << endl;

auto ctAND1 = cc.EvalBinGate(AND, bit1, bit2);
auto bit2Not = cc.EvalNOT(bit2);
auto ctAND2 = cc.EvalBinGate(AND, bit2Not, bit1);
auto ctResult = cc.EvalBinGate(OR, ctAND1, ctAND2);

LWEPlaintext result;

cc.Decrypt(sk, ctResult, &result);

printf("b1=%d\n",b1);
printf("b2=%d\n",b2);
printf("(b1 AND b2) OR ( b1 AND NOT(b2))\n");
printf("(%d AND %d) OR ( %d AND NOT(%d))=%d\n",b1,b2,b1,b2,result);
```

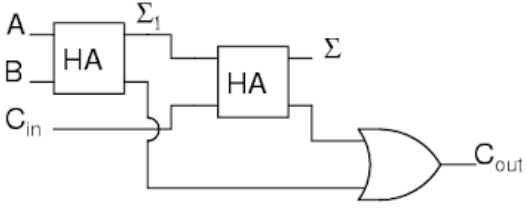
https://asecuritysite.com/openfhe/openfhe_09cpp

https://asecuritysite.com/openfhe/openfhe_09cpp_pke

MD/FHEW








```
    cout <<"Val1="<< val1 << " Binary: "<< bin1[3] << bin1[2] << bin1[1] << bin1[0] << endl;
    cout <<"Val2="<< val2 << " Binary: "<< bin2[3] << bin2[2] << bin2[1] << bin2[0] << endl;

    auto bin1_0 = cc.Encrypt(sk, bin1[0]);
     auto bin1_1 = cc.Encrypt(sk, bin1[1]);
   auto bin1_2 = cc.Encrypt(sk, bin1[2]);
    auto bin1_3 = cc.Encrypt(sk, bin1[3]);

        auto bin2_0 = cc.Encrypt(sk, bin2[0]);
     auto bin2_1 = cc.Encrypt(sk, bin2[1]);
   auto bin2_2 = cc.Encrypt(sk, bin2[2]);
    auto bin2_3 = cc.Encrypt(sk, bin2[3]);

        auto c_carryin = cc.Encrypt(sk, 0);


.WECiphertext c_sum1,c_carryout,c_sum2,c_sum3,c_sum4;

    tie(c_sum1,c_carryout)=FA(cc,bin1_0,bin2_0,c_carryin );
    tie(c_sum2,c_carryout)=FA(cc,bin1_1,bin2_1,c_carryout );
    tie(c_sum3,c_carryout)=FA(cc,bin1_2,bin2_2,c_carryout );
    tie(c_sum4,c_carryout)=FA(cc,bin1_3,bin2_3,c_carryout );
```

https://asecuritysite.com/openfhe/openfhe_11cpp
https://asecuritysite.com/openfhe/openfhe_11cpp_pke

Chebyshev Functions

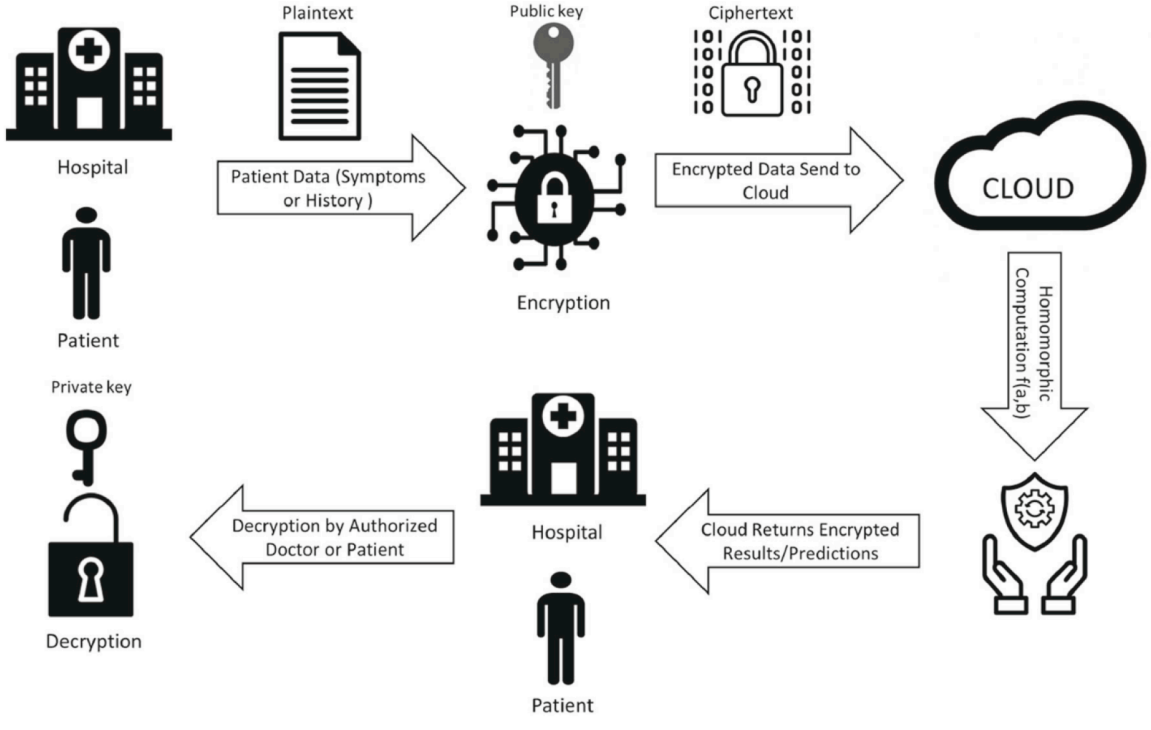
With approximation theory, it is possible to determine an approximate polynomial p(x) that is approximate to a function f(x).

$$
\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1 \\
T_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x \\
T_8(x) &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\
T_9(x) &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x \\
T_{10}(x) &= 512x^{10} - 1280x^8 + 1120x^6 - 400x^4 + 50x^2 - 1 \\
T_{11}(x) &= 1024x^{11} - 2816x^9 + 2816x^7 - 1232x^5 + 220x^3 - 11x
\end{aligned}
$$

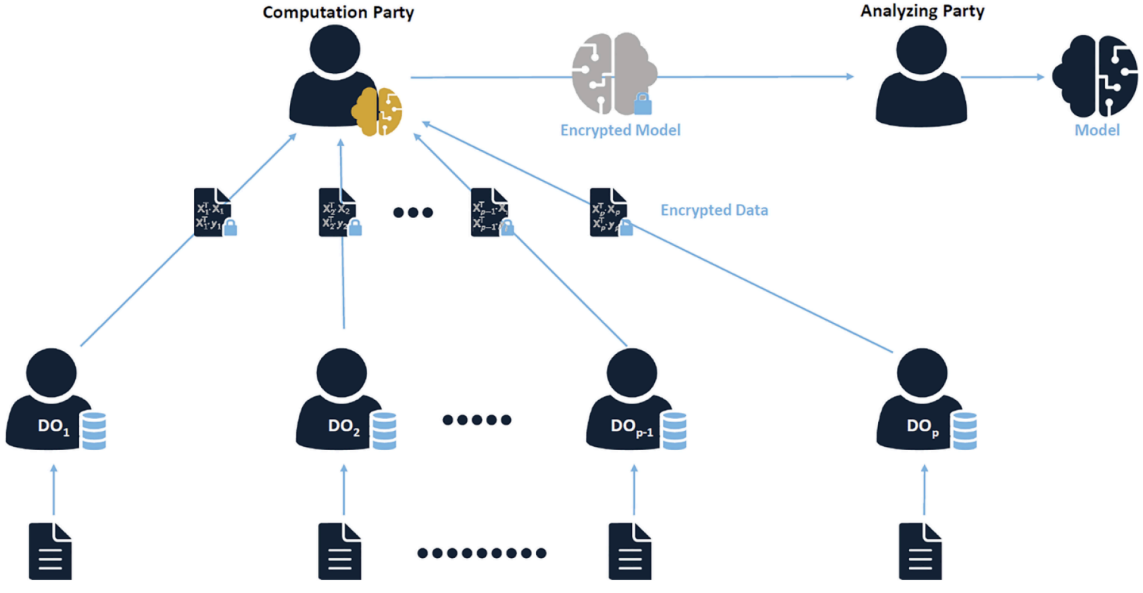
```
 if (opt==0) {
     result = cc->EvalChebyshevFunction([](double x) -> double { return std::log10(x); }, cipherte
     std::cout <<" x     log10(x)\n----------" << std::endl;
 }
else if (opt==1) {
 result = cc->EvalChebyshevFunction([](double x) -> double { return std::log2(x); }, ciphertext, l
  std::cout <<" x     log2(x)\n----------" << std::endl;
}
else if (opt==2) {
 result = cc->EvalChebyshevFunction([](double x) -> double { return std::log(x); }, ciphertext, lo
    std::cout <<" x     ln(x)\n----------" << std::endl;
}
   else if (opt==3) {
     result = cc->EvalChebyshevFunction([](double x) -> double { return std::exp(x); }, ciphertext
    std::cout <<" x     exp(x)\n----------" << std::endl;
}
else if (opt==4) {
     result = cc->EvalChebyshevFunction([](double x) -> double { return std::exp2(x); }, ciphertex
    std::cout <<" x     2^x\n----------" << std::endl;
}
```

https://asecuritysite.com/openfhe/openfhe_18cpp

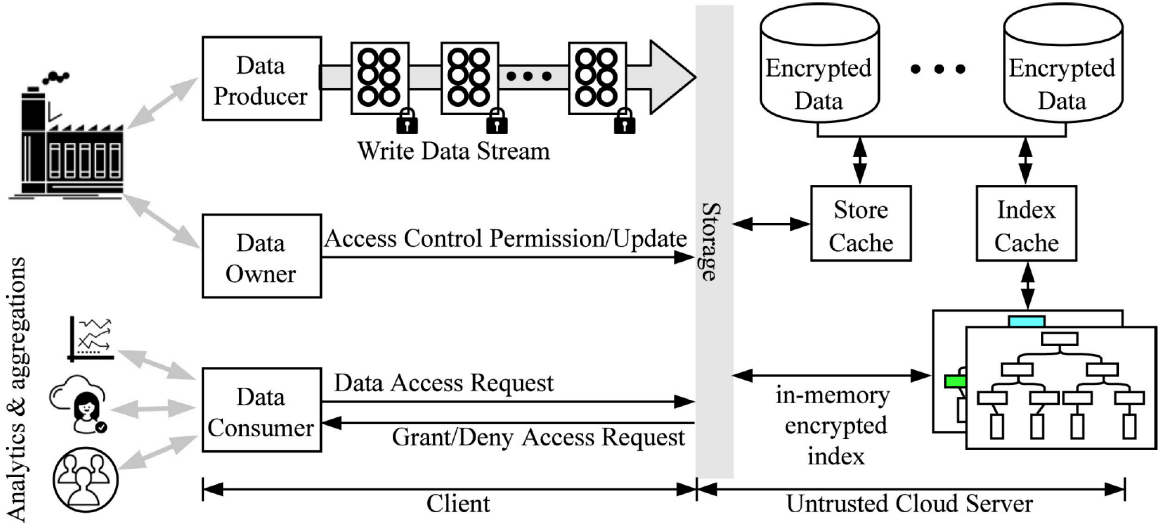
Homomorphic Sharing



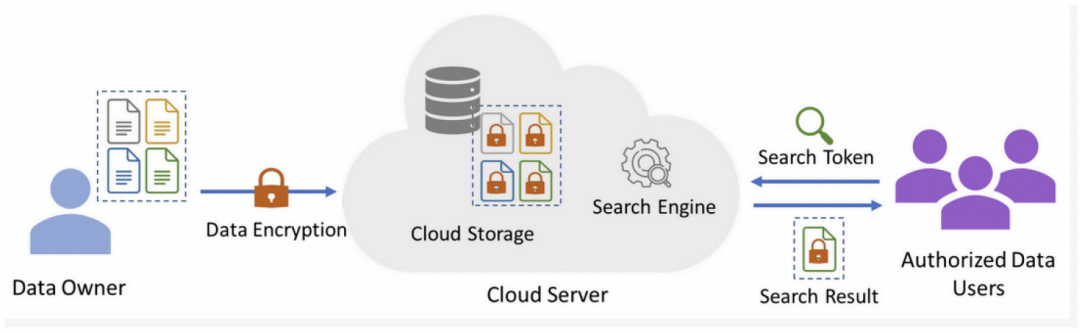
Multiparty Sharing

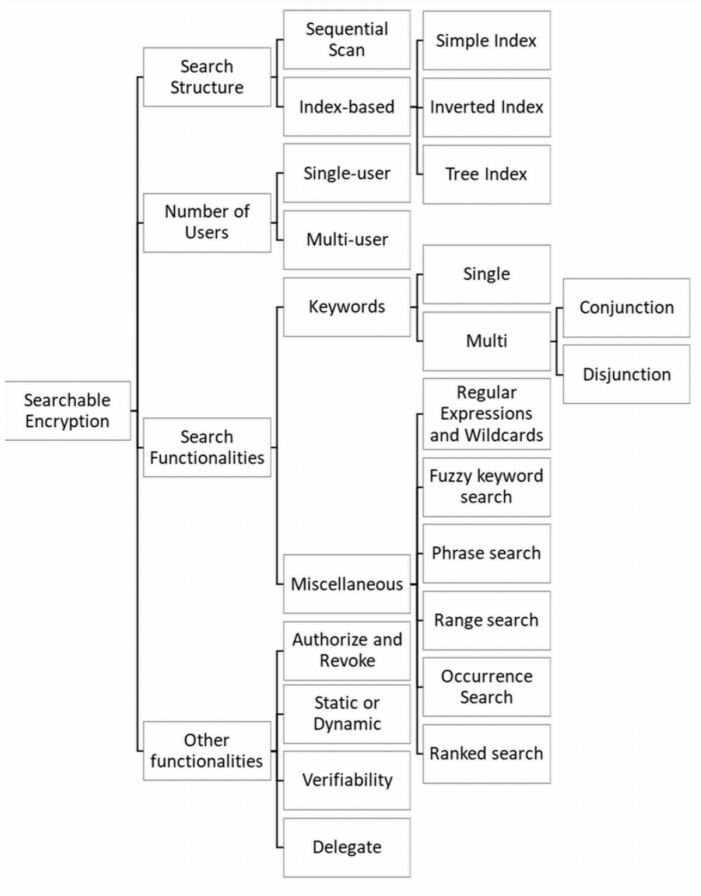


SmartCrypt


Data Producer
Write Data Stream
Encrypted Data
Encrypted Data
Data Owner
Access Control Permission/Update
Storage
Store Cache
Index Cache
Analytics & aggregations
Data Consumer
Data Access Request
Grant/Deny Access Request
in-memory encrypted index
Client
Untrusted Cloud Server

Searchable Encryption

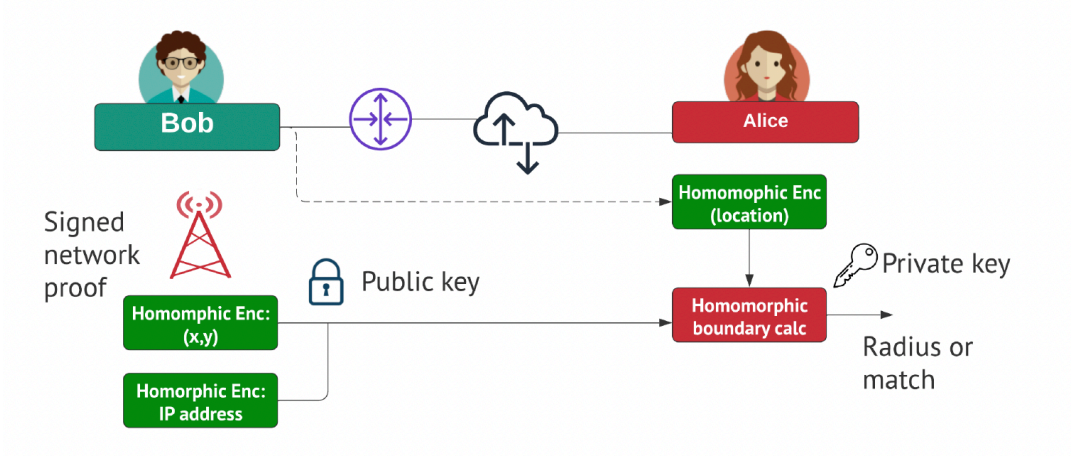




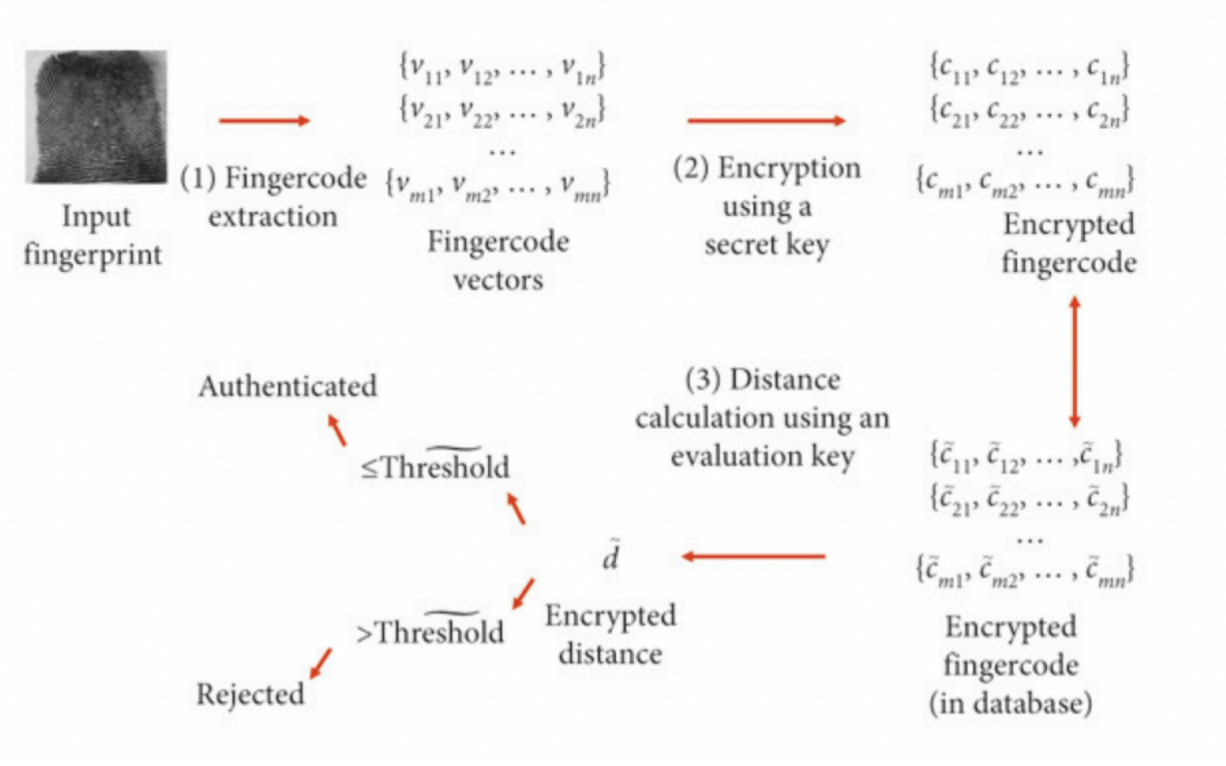
Location Tracking

```
E(TIME1)a E(Location1)a
E(TIME2)a E(Location2)a
E(TIME3)a E(Location2)a
E(TIME1)b E(Location1)b
E(TIME2)b E(Location2)b
E(TIME3)b E(Location2)b
```

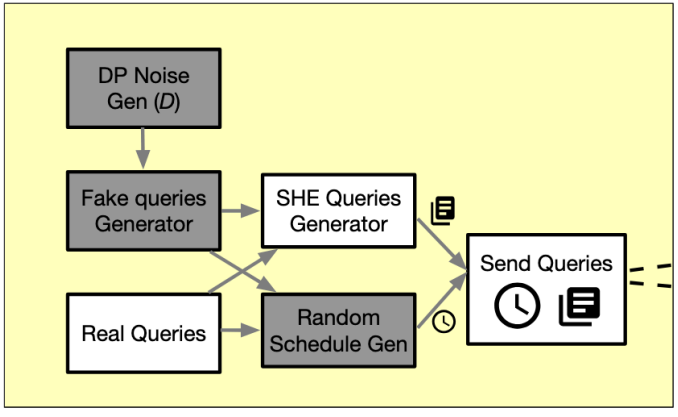
Timestamp	u_1 $rssi$	m	Type	u_2 $rssi$
1509240563.03	-64	D8:84:66:4C:D1:00	WiFi	-85
1509240563.03	-69	D8:84:66:4E:E4:F0	WiFi	-79
1509240563.03	-59	D8:84:66:4E:F0:04	BL	-91

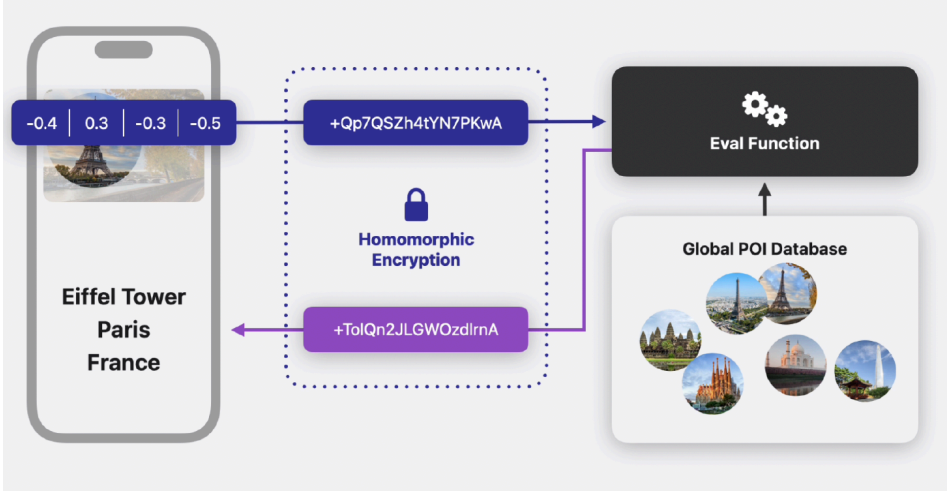


Privacy-aware biometrics/ID



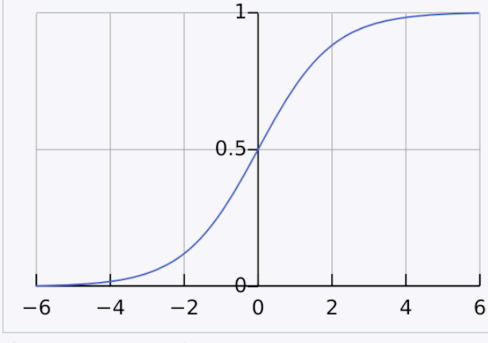
Apple Privacy Aware





Logistic Function

With homomorphic encryption we can represent a mathematical operation in the form for a homomorphic equation. One of the most widely used methods is to use Chebyshev polynomials, and which allows the mapping of the function to a Chebyshev approximation. In this case, we will use homomorphic encryption to approximate a logistic function (and which is represented by f(x)=1/(1+e^{-x}).



Standard logistic function where $L = 1, k = 1, x_0 = 0$.

$$f(x) = \frac{1}{1+e^{-x}}$$

```
std::cout << "Logistic Evaluation \n" << std::endl;

CCParams<CryptoContextCKKSRNS> parameters;
parameters.SetMultiplicativeDepth(5);
parameters.SetScalingModSize(40);

CryptoContext<DCRTPoly> cc = GenCryptoContext(parameters);
cc->Enable(PKE);
cc->Enable(KEYSWITCH);
cc->Enable(LEVELEDSHE);
cc->Enable(ADVANCEDSHE);

size_t encodedLength = input.size();

Plaintext plaintext1 = cc->MakeCKKSPackedPlaintext(input);

auto keyPair = cc->KeyGen();

std::cout << "Generating evaluation key.";
cc->EvalMultKeyGen(keyPair.secretKey);

auto ciphertext1 = cc->Encrypt(keyPair.publicKey, plaintext1);

auto result = cc->EvalLogistic(ciphertext1,-1,1,3);

Plaintext plaintextDec;

cc->Decrypt(keyPair.secretKey, result, &plaintextDec);

plaintextDec->SetLength(encodedLength);
```

https://asecuritysite.com/openfhe/openfhe_19cpp

Matrix Operations

The inner product of two vectors of a and b is represented by $\langle a, b \rangle$. It is the dot product of two vectors, and represented as $\langle a, b \rangle = a.ycos(\theta)$, and where θ is the angle between the two vectors.

If we have a vector of x=(10,20,15), then the magniture will be:

$$a = \sqrt{10^2 + 20^2 + 15^2} = 26.93$$

If we have the same vector of b=(10,20,15), we will have the same magnitude. The inner product will then be:

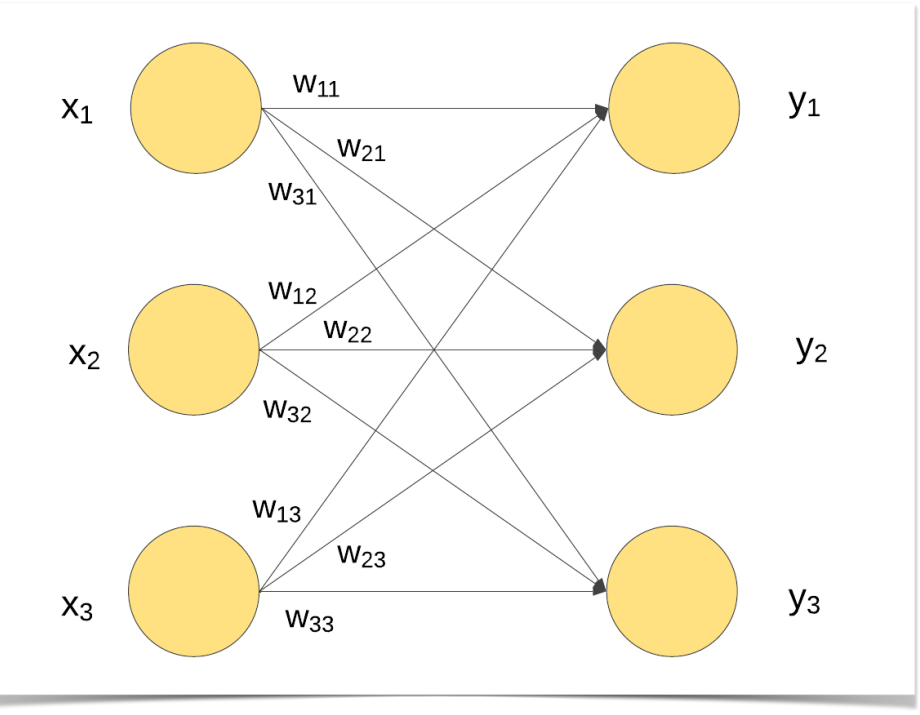
$$\langle a, b \rangle = |a|.|b|.cos(\theta) = 26.93 \times 26.93.cos(0) = 725$$

```
lbcrypto::CryptoContext<lbcrypto::DCRTPoly> cc;
cc = GenCryptoContext(parameters);

cc->Enable(PKE);
cc->Enable(LEVELEDSHE);
cc->Enable(ADVANCEDSHE);

KeyPair keys = cc->KeyGen();
cc->EvalMultKeyGen(keys.secretKey);
cc->EvalSumKeyGen(keys.secretKey);

Plaintext plaintext1 = cc->MakeCKKSPackedPlaintext(v1);
auto ct1             = cc->Encrypt(keys.publicKey, plaintext1);

Plaintext plaintext2 = cc->MakeCKKSPackedPlaintext(v2);
auto ct2             = cc->Encrypt(keys.publicKey, plaintext2);

auto finalResult     = cc->EvalInnerProduct(ct1, ct2, batchSize);
lbcrypto::Plaintext res;
cc->Decrypt(keys.secretKey, finalResult, &res);
res->SetLength(v1.size());
auto final = res->GetCKKSPackedValue()[0].real();

    std::cout <<  "v1=" << s1 << std::endl;
        std::cout <<  "v2=" << s2 << std::endl;
std::cout <<  "Inner Product Result: " << final << std::endl;
std::cout << "Expected value: " << inner_product(v1.begin(), v1.end(), v2.begin(), 0) << std::end
```

https://asecuritysite.com/openfhe/openfhe_13cpp

https://asecuritysite.com/openfhe/openfhe_14cpp

Matrix Operations



If we have a vector of the form:

$$v_1 = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}$$

and a matrix of:

$$m_1 = \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{bmatrix}$$

We now get:

$$v_1 . m_1 = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{bmatrix}$$

and:

$$v_1 . m_1 = \begin{bmatrix} x_1 . w_{11} + x_2 . w_{21} + x_3 . w_{31} & x_1 . w_{12} + x_2 . w_{22} + x_3 . w_{32} & x_1 . w_{13} + x_2 . w_{23} + x_3 . w_{33} \end{bmatrix}$$

Thus we get:

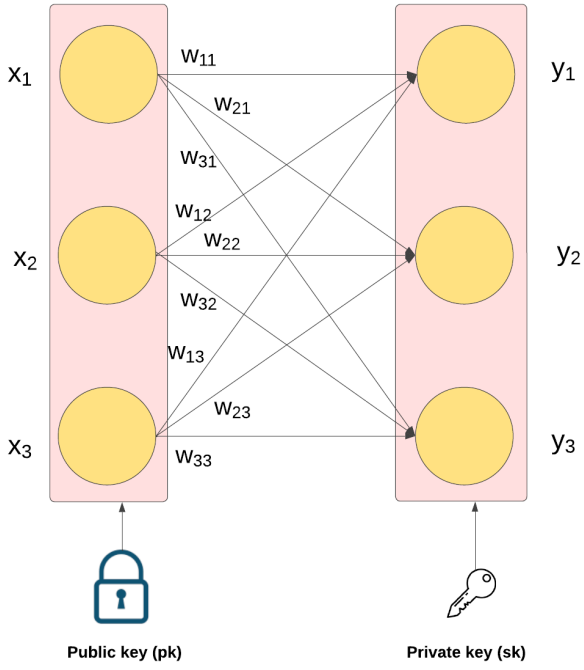
$$y_1 = x_1 . w_{11} + x_2 . w_{21} + x_3 . w_{31}$$

$$y_2 = x_1 . w_{12} + x_2 . w_{22} + x_3 . w_{32}$$

$$y_3 = x_1 . w_{13} + x_2 . w_{23} + x_3 . w_{33}$$

https://asecuritysite.com/openfhe/openfhe_27cpp

Matrix Operations



$$v_1 . m_1 = \left[Ek_{pk}(x_1) \quad Ek_{pk}(x_2) \quad Ek_{pk}(x_3) \right] \begin{bmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \\ w_{13} & w_{23} & w_{33} \end{bmatrix}$$

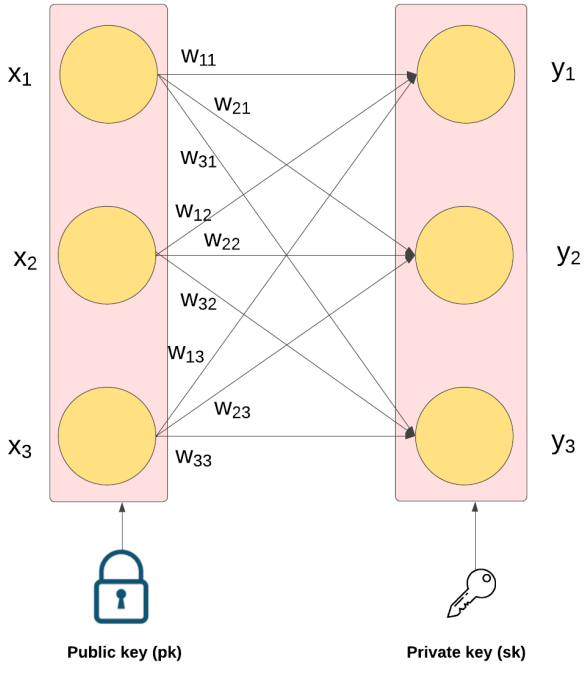
We can then use the private key to determine:

$$\left[y_1 \quad y_2 \quad y_3 \right] = Ek_{sk}^{-1}(v_1 . m_1)$$

https://asecuritysite.com/openfhe/openfhe_27cpp

https://asecuritysite.com/openfhe/openfhe_26cpp

Matrix Operations

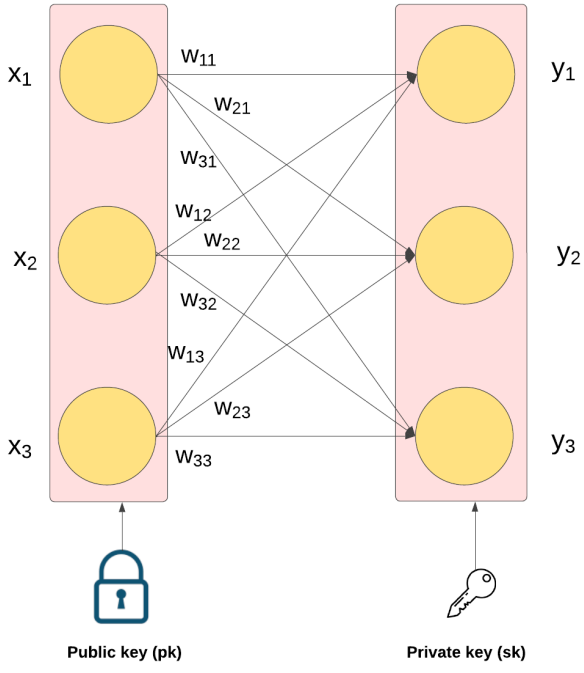



```
std::cout << "Generating rotation keys... ";
std::vector<int32_t> rotationKeys = {};
for (int i = -ROWS*COLS; i <= ROWS*COLS; i++) rotationKeys.push_back(i);
cryptoContext->EvalRotateKeyGen(keyPair.secretKey, rotationKeys);
std::cout << "Done"<< std::endl << std::endl;

std::vector<double> vector = genRandVect(ROWS, max,0);
Plaintext vectorP  = cryptoContext->MakeCKKSPackedPlaintext(vector);

std::vector<std::vector<double>> matrix = genRandMatrix(ROWS, COLS, max,1);

std::cout << "Vector (V1) = " << vector << std::endl;
std::cout << "Matrix (M1) = " << matrix << std::endl;

Ciphertext<DCRTPoly> vectorC = cryptoContext->Encrypt(keyPair.publicKey, vectorP);

Ciphertext<DCRTPoly> resC;
Plaintext res;
std::vector<double> resOutput, resOutputtmp;

resOutput = vectorMatrixMult(vector, matrix);

std::cout << "V1*M1   = " << resOutput << std::endl;

resC = vectorMatrixMultByInnProdCP(cryptoContext, keyPair.publicKey, vectorC, matrix);

cryptoContext->Decrypt(keyPair.secretKey, resC, &res);
res->SetLength(COLS);

std::cout << "vectorC * matrix (by inner product)      = " << res->GetCKKSPackedValue()  << std::en
```

https://asecuritysite.com/openfhe/openfhe_27cpp

https://asecuritysite.com/openfhe/openfhe_26cpp

Matrix Operations

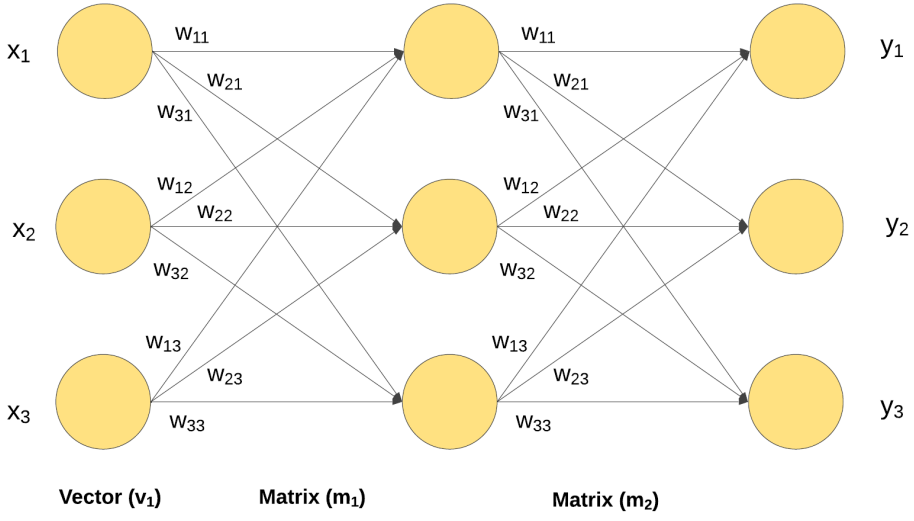



```
Ciphertext<DCRTPoly> vectorC = cryptoContext->Encrypt(keyPair.publicKey, vectorP);

Ciphertext<DCRTPoly> resC;
Plaintext res;
std::vector<int64_t> resOutput, resOutputtmp;


 resOutput = vectorMatrixMult(vector, matrix1);
resOutput = vectorMatrixMult(resOutput, matrix2);

std::cout << "V1*M1*M2 (non homomorphic)  = " << resOutput << std::endl;

//resC = vectorMatrixMultByInnProdCP(cryptoContext, keyPair.publicKey, vectorC, matrix1);
//resC = vectorMatrixMultByInnProdCP(cryptoContext, keyPair.publicKey, resC, matrix2);
resC = vectorMatrixMultByInnProdFastCP(cryptoContext, keyPair.publicKey, vectorC, matrix1);
 resC = vectorMatrixMultByInnProdFastCP(cryptoContext, keyPair.publicKey, resC, matrix2);

cryptoContext->Decrypt(keyPair.secretKey, resC, &res);
res->SetLength(n3);
resOutput = res->GetPackedValue();
std::cout << "V1*M1*M3 (by inner product)      = " << resOutput  << std::endl;
```

https://asecuritysite.com/openfhe/openfhe_27cpp

https://asecuritysite.com/openfhe/openfhe_26cpp

Matrix Operations

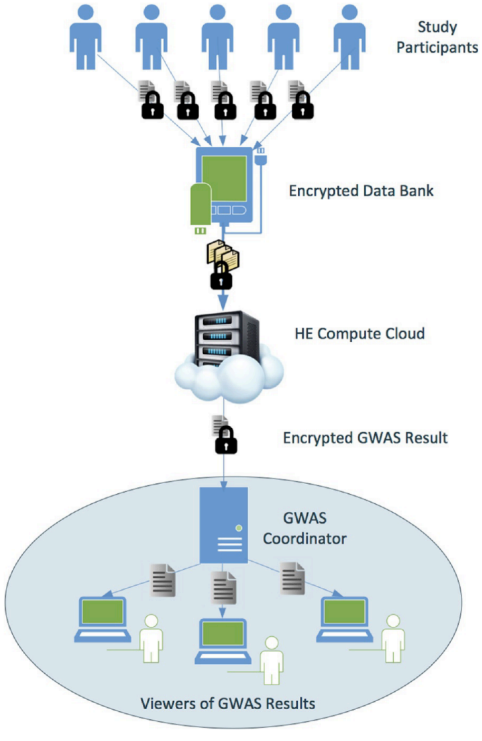



```
Ciphertext<DCRTPoly> vectorC = cryptoContext->Encrypt(keyPair.publicKey, vectorP);

Ciphertext<DCRTPoly> resC;
Plaintext res;
std::vector<int64_t> resOutput, resOutputtmp;


 resOutput = vectorMatrixMult(vector, matrix1);
resOutput = vectorMatrixMult(resOutput, matrix2);

std::cout << "V1*M1*M2 (non homomorphic)  = " << resOutput << std::endl;

//resC = vectorMatrixMultByInnProdCP(cryptoContext, keyPair.publicKey, vectorC, matrix1);
//resC = vectorMatrixMultByInnProdCP(cryptoContext, keyPair.publicKey, resC, matrix2);
resC = vectorMatrixMultByInnProdFastCP(cryptoContext, keyPair.publicKey, vectorC, matrix1);
 resC = vectorMatrixMultByInnProdFastCP(cryptoContext, keyPair.publicKey, resC, matrix2);

cryptoContext->Decrypt(keyPair.secretKey, resC, &res);
res->SetLength(n3);
resOutput = res->GetPackedValue();
std::cout << "V1*M1*M3 (by inner product)      = " << resOutput  << std::endl;
```

https://asecuritysite.com/openfhe/openfhe_28cpp

https://asecuritysite.com/openfhe/openfhe_29cpp

GWAS (Gnome-wide Association Studies)



	GLM		HE LRA		HE Chisq	
SNP	OR	stat	OR	stat	OR	stat
rs10033900_T	1.09	1.97	1.08	1.91	1.06	1.44
rs943080_C	0.88	−2.94	0.89	−2.88	0.91	−2.26
rs79037040_G	0.88	−2.98	0.88	−2.91	0.89	−2.82
rs2043085_T	0.91	−2.01	0.92	−1.95	0.92	−2.13
rs2230199_C	1.41	6.83	1.38	6.67	1.40	7.10
rs8135665_T	1.12	2.04	1.12	2.03	1.12	2.29
rs114203272_T	0.62	−3.55	0.63	−3.50	0.67	−3.08
rs114212178_T	0.87	−0.70	0.87	−0.69	0.86	−0.77

Figure 7.8: Results for GWAS [92]

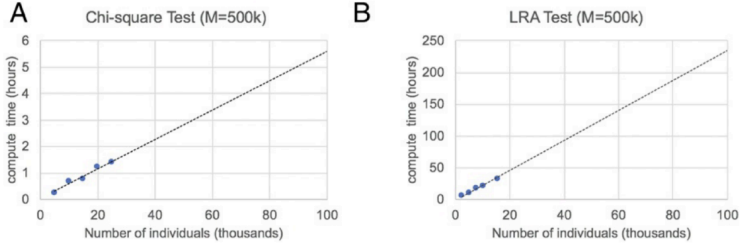
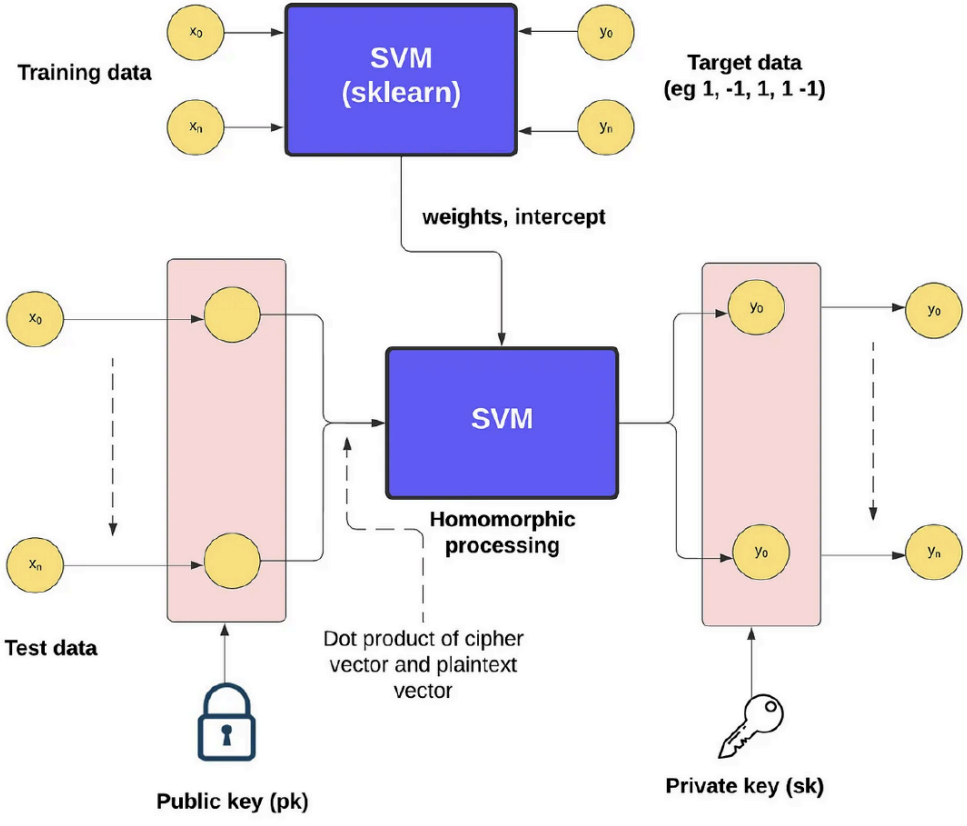


Figure 7.9: Results for GWAS [92]

SVM


Training data
SVM (sklearn)
Target data (eg 1, -1, 1, 1 -1)
weights, intercept
SVM
Homomorphic processing
Test data
Dot product of cipher vector and plaintext vector
Public key (pk)
Private key (sk)

SVM

Evaluation of Privacy-aware Support Vector Machine (SVM) Learning using Homomorphic Encryption

William J Buchanan[1] and Hisham Ali[1]

Blockpass ID Lab, Edinburgh Napier University
b.buchanan@napier.ac.uk

Abstract. The requirement for privacy-aware machine learning increases as we continue to use PII (Personally Identifiable Information) within machine training. To overcome these privacy issues, we can apply Fully Homomorphic Encryption (FHE) to encrypt data before it is fed into a machine learning model. This involves creating a homomorphic encryption key pair, and where the associated public key will be used to encrypt the input data, and the private key will decrypt the output. But, there is often a performance hit when we use homomorphic encryption, and so this paper evaluates the performance overhead of using the SVM machine learning technique with the OpenFHE homomorphic encryption library. This uses Python and the scikit-learn library for its implementation. The experiments include a range of variables such as multiplication depth, scale size, first modulus size, security level, batch size, and ring dimension, along with two different SVM models, SVM-Poly and SVM-Linear. Overall, the results show that the two main parameters which affect performance are the ring dimension and the modulus size, and that SVM-Poly and SVM-Linear show similar performance levels.

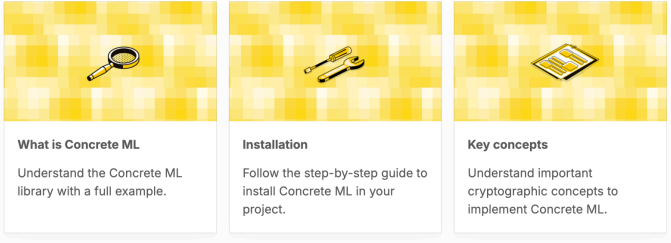
Concrete-ML

Welcome

Concrete ML is an open-source, privacy-preserving, machine learning framework based on Fully Homomorphic Encryption (FHE).

Get started

Learn the basics of Concrete ML, set it up, and make it run with ease.



What is Concrete ML

Understand the Concrete ML library with a full example.

Installation

Follow the step-by-step guide to install Concrete ML in your project.

Key concepts

Understand important cryptographic concepts to implement Concrete ML.

Concrete-ML also allows for a direct replacement for the following methods:

- Linear models: LinearRegression, LogisticRegression, LinearSVC, LinearSVR PoissonRegressor, TweedieRegressor, GammaRegressor, Lasso, Ridge, ElasticNet and SGDRegressor.

- Tree Models: DecisionTreeClassifier, DecisionTreeRegressor, RandomForestClassifier, and RandomForestRegressor.
- NeuralNetClassifier (MLPClassifier) and NeuralNetRegressor (MLPRegressor).
- Nearest Neighbour. KNeighborsClassifier

GPUs (BFV)

Feature	CPU	GPU	
		RTX3060Ti	GTX1080
Model	Ryzen7 3800X	RTX3060Ti	GTX1080
Threads	16	4864	2560
Freq.	4.20 GHz	1665 MHz	1733 MHz
RAM	32 GB (3600 MHz)	8 GB	8 GB
Mem. Type	-	GDDR6	GDDR5X
Mem. Bus	-	256 bits	256 bits
Bandwidth	-	448 GB/s	320 GB/s

		CPU		GPU	
n	**Count**	**Power***	**Time**	**Power***	**Time**
2^{12}	1	55.5 W	3025 $\boldsymbol{\mu s}$	44.37 W	92 μs
	10	60.49 W	30430 $\boldsymbol{\mu s}$	44.5 W	830 μs
	100	60.65 W	317682 μs	44.54 W	9890 μs
	500	63.55 W	1369910 μs	44.49 W	51246 μs
2^{13}	1	55.71 W	9121 μs	44.15 W	161 μs
	10	59.52 W	89266 $\boldsymbol{\mu s}$	44.25 W	1589 μs
	100	61.39 W	852098 μs	44.24 W	16275 μs
	500	64.78 W	3970960 μs	47.05 W	81714 μs
2^{14}	1	57.23 W	38414 μs	44.35 W	829 μs
	10	61.31 W	381921 $\boldsymbol{\mu s}$	44.37 W	7515 μs
	100	60.8 W	3763901μs	44.61 W	71858 μs
	500	64.16 W	18786647 μs	48.60 W	337236 μs
2^{15}	1	59.55 W	186334 μs	44.29 W	3382 μs
	10	59.46 W	1796976 $\boldsymbol{\mu s}$	45.48 W	36301 μs
	100	60.89 W	17822724 μs	55.71 W	338212 μs
	500	66.93 W	89749372 μs	65.13 W	342199 μs

			GPU with [29] NTT		GPU with new NTT		[22]	SEAL	T
Operation	**n**	$\log_2 q$	RTX3060Ti	GTX1080	RTX3060Ti	GTX1080	Tesla V100	CPU	T_s
Add.	2^{12}	109	4 μs	4.6 μs	4 μs	4.6 μs	-	14 μs	3.5×
	2^{13}	218	5.1μs	6.2 μs	5.1 μs	6.1 μs	-	58 μs	11.37×
	2^{14}	438	**12.3 $\boldsymbol{\mu s}$**	19.4 μs	**12.3 $\boldsymbol{\mu s}$**	19.4 μs	-	**233 $\boldsymbol{\mu s}$**	**18.94**×
	2^{15}	881	44 μs	64.2 μs	44 μs	64.2 μs	-	778 μs	17.68×
Mult.	2^{12}	109	172 μs	259 μs	86 μs	155.8 μs	-	3212 μs	37.3×
	2^{13}	218	297 μs	532 μs	202 μs	423.4 μs	-	11883 μs	58.8×
	2^{14}	438	**1037 $\boldsymbol{\mu s}$**	2294 μs	**768 $\boldsymbol{\mu s}$**	1856.1 μs	-	**48757 $\boldsymbol{\mu s}$**	**63.4**×
	2^{15}	881	5372 μs	10657 μs	3757 μs	-	-	205295 μs	54.6×
Relin.	2^{12}	109	46 μs	82.7 μs	39.51 μs	59.3 μs	-	625 μs	15.81×
	2^{13}	218	104 μs	145 μs	88.54 μs	143.4 μs	-	3100 μs	35.01×
	2^{14}	438	**462 $\boldsymbol{\mu s}$**	1013 μs	**376.61 $\boldsymbol{\mu s}$**	825.3 μs	-	**18295 $\boldsymbol{\mu s}$**	**48.57**×
	2^{15}	881	3530 μs	6651 μs	3150 μs	-	-	111736 μs	35.47×
Rot.	2^{12}	109	51 μs	87 μs	42.1 μs	59.4 μs	-	642 μs	15.24×
	2^{13}	218	116 μs	172 μs	103.3 μs	162.7 μs	-	3157 μs	30.56×
	2^{14}	438	**544 $\boldsymbol{\mu s}$**	1339 μs	**458.7 $\boldsymbol{\mu s}$**	1067.2 μs	-	**18338 $\boldsymbol{\mu s}$**	**39.97**×
	2^{15}	881	3879 μs	10504 μs	3464.5 μs	-	-	113437 μs	32.74×
Mult. + Relin.	2^{12}	60	-	-	**136 $\boldsymbol{\mu s}$**	-	**859 $\boldsymbol{\mu s}$**	-	**6.31**×
	2^{13}	120	-	-	170 μs	-	1012 μs	-	5.95×
	2^{14}	360	-	-	661 μs	-	2010 μs	-	3.04×
	2^{15}	600	-	-	2875 μs	-	4826 μs	-	1.67×

GPUs (BFV)

Feature	CPU	GPU	
		RTX3060Ti	GTX1080
Model	Ryzen7 3800X	RTX3060Ti	GTX1080
Threads	16	4864	2560
Freq.	4.20 GHz	1665 MHz	1733 MHz
RAM	32 GB (3600 MHz)	8 GB	8 GB
Mem. Type	-	GDDR6	GDDR5X
Mem. Bus	-	256 bits	256 bits
Bandwidth	-	448 GB/s	320 GB/s

n	Count	CPU		GPU	
		Power*	Time	Power*	Time
2^{12}	1	55.5 W	3025 μs	44.37 W	92 μs
	10	60.49 W	30430 μs	44.5 W	830 μs
	100	60.65 W	317682 μs	44.54 W	9890 μs
	500	63.55 W	1369910 μs	44.49 W	51246 μs
2^{13}	1	55.71 W	9121 μs	44.15 W	161 μs
	10	59.52 W	89266 μs	44.25 W	1589 μs
	100	61.39 W	852098 μs	44.24 W	16275 μs
	500	64.78 W	3970960 μs	47.05 W	81714 μs
2^{14}	1	57.23 W	38414 μs	44.35 W	829 μs
	10	61.31 W	381921 μs	44.37 W	7515 μs
	100	60.8 W	3763901μs	44.61 W	71858 μs
	500	64.16 W	18786647 μs	48.60 W	337236 μs
2^{15}	1	59.55 W	186334 μs	44.29 W	3382 μs
	10	59.46 W	1796976 μs	45.48 W	36301 μs
	100	60.89 W	17822724 μs	55.71 W	338212 μs
	500	66.93 W	89749372 μs	65.13 W	342199 μs

Operation	n	$\log_2 q$	GPU with [29] NTT		GPU with new NTT		[22]	SEAL	T
			RTX3060Ti	GTX1080	RTX3060Ti	GTX1080	Tesla V100	CPU	T_s
Add.	2^{12}	109	4 μs	4.6 μs	4 μs	4.6 μs	-	14 μs	3.5×
	2^{13}	218	5.1μs	6.2 μs	5.1 μs	6.1 μs	-	58 μs	11.37×
	2^{14}	438	**12.3 μs**	19.4 μs	**12.3 μs**	19.4 μs	-	**233 μs**	**18.94**×
	2^{15}	881	44 μs	64.2 μs	44 μs	64.2 μs	-	778 μs	17.68×
Mult.	2^{12}	109	172 μs	259 μs	86 μs	155.8 μs	-	3212 μs	37.3×
	2^{13}	218	297 μs	532 μs	202 μs	423.4 μs	-	11883 μs	58.8×
	2^{14}	438	**1037 μs**	2294 μs	**768 μs**	1856.1 μs	-	**48757 μs**	**63.4**×
	2^{15}	881	5372 μs	10657 μs	3757 μs	-	-	205295 μs	54.6×
Relin.	2^{12}	109	46 μs	82.7 μs	39.51 μs	59.3 μs	-	625 μs	15.81×
	2^{13}	218	104 μs	145 μs	88.54 μs	143.4 μs	-	3100 μs	35.01×
	2^{14}	438	**462 μs**	1013 μs	**376.61 μs**	825.3 μs	-	**18295 μs**	**48.57**×
	2^{15}	881	3530 μs	6651 μs	3150 μs	-	-	111736 μs	35.47×
Rot.	2^{12}	109	51 μs	87 μs	42.1 μs	59.4 μs	-	642 μs	15.24×
	2^{13}	218	116 μs	172 μs	103.3 μs	162.7 μs	-	3157 μs	30.56×
	2^{14}	438	**544 μs**	1339 μs	**458.7 μs**	1067.2 μs		**18338 μs**	**39.97**×
	2^{15}	881	3879 μs						
Mult. + Relin.	2^{12}	60	-						
	2^{13}	120	-						
	2^{14}	360	-						
	2^{15}	600	-						

n	NTT_count	32 Bit (Implemented On RTX 3060Ti)						64 Bit (Implemented On RTX 3060Ti)					
		Forward NTT			Inverse NTT			Forward NTT			Inverse NTT		
		[29]	T.W.	T	[29]	T.W.	T	[29]	T.W.	T	[29]	T.W.	T
2^{12}	4	12.3 μs	**11 μs**	**1.12**×	11.2 μs	**11.1 μs**	**1.11**×	19.5 μs	**14.3 μs**	**1.36**×	16 μs	**15.4 μs**	**1.03**×
	16	13 μs	**11.2μs**	**1.16**×	12.2 μs	12.2 μs	1×	22.5 μs	**17.2 μs**	**1.30**×	17.8 μs	19.4 μs	0.91×
	32	13.9 μs	17.9 μs	0.77 ×	18.4 μs	19.2 μs	0.96×	23.5 μs	25.2 μs	0.93×	29.3 μs	**26.6 μs**	**1.10**×
	64	23.1 μs	28.6 μs	0.80×	29.5 μs	**29.3 μs**	**1**×	39.9 μs	43 μs	0.93×	52.9 μs	**50.1 μs**	**1.05**×
	128	38.9 μs	46.7 μs	0.83×	48.1 μs	48.7 μs	0.98×	75.7 μs	81.6 μs	0.92×	96.5 μs	**91.1 μs**	**1.06**×
2^{13}	4	17.1 μs	**12.2 μs**	**1.40**×	14.1 μs	14.3 μs	0.98×	24.4 μs	**16.6 μs**	**1.47**×	21.1 μs	**20.5 μs**	**1.03**×
	16	18.4 μs	18.4 μs	**1**×	22.2 μs	**20.3 μs**	**1.09**×	28.2 μs	**25.6 μs**	**1.10**×	34.8 μs	**28.6 μs**	**1.21**×
	32	28.6 μs	29.4 μs	0.97×	34.9 μs	**30.3 μs**	**1.15**×	47.9 μs	**44.4 μs**	**1.08**×	59.4 μs	**49.9 μs**	**1.19**×
	64	49.8 μs	**46.7 μs**	**1.07**×	57.3 μs	**50.7 μs**	**1.13**×	97.1 μs	**82.1 μs**	**1.18**×	121.2 μs	**93.9 μs**	**1.29**×
	128	88 μs	91.2 μs	0.96×	124 μs	**96.1 μs**	**1.29**×	170.7 μs	**156.4 μs**	**1.09**×	224.1 μs	**173.6 μs**	**1.29**×
2^{14}	4	20.1 μs	**15.2 μs**	**1.32**×	17.6 μs	**16.3 μs**	**1.08**×	29.98 μs	**21.76 μs**	**1.37**×	24.5 μs	**24.1 μs**	**1.01**×
	16	33.7 μs	**30.5 μs**	**1.05**×	40.9 μs	**30.1 μs**	**1.36**×	54.4 μs	**46.54 μs**	**1.17**×	65.9 μs	**53.2 μs**	**1.23**×
	32	57.3 μs	**50.1 μs**	**1.14**×	64.5 μs	**51.8 μs**	**1.24**×	121.8 μs	**84.6 μs**	**1.44**×	143.3 μs	**97.2 μs**	**1.47**×
	64	112.6 μs	**96 μs**	**1.17**×	147.4 μs	**99.3 μs**	**1.48**×	218.5 μs	**160.5 μs**	**1.36**×	266.8 μs	**180 μs**	**1.48**×
	128	210.7 μs	**176.1 μs**	**1.19**×	277.1 μs	**183 μs**	**1.51**×	420.8 μs	**303.19 μs**	**1.38**×	511.9 μs	**331.7 μs**	**1.54**×
2^{15}	4	25.6 μs	26.4 μs	0.96×	28.7 μs	**25.9 μs**	**1.10**×	35.8 μs	41.2 μs	0.86×	47.3 μs	50.3 μs	0.94×
	16	64.1 μs	**52.2 μs**	**1.22**×	74.7 μs	**53.2 μs**	**1.40**×	147.1 μs	**100 μs**	**1.47**×	170.1 μs	**95.6 μs**	**1.78**×
	32	136.3 μs	**100.3 μs**	**1.35**×	173 μs	**102.4 μs**	**1.69**×	266.2 μs	**191.8 μs**	**1.38**×	325.5 μs	**193.2 μs**	**1.68**×
	64	254.9 μs	**192.1 μs**	**1.32**×	322.2 μs	**193.6 μs**	**1.66**×	514.2 μs	**372.2 μs**	**1.38**×	633.7 μs	**377.7 μs**	**1.67**×
	128	491.1 μs	**362.4 μs**	**1.35**×	623.1 μs	**364.3 μs**	**1.71**×	998.9 μs	**709.3 μs**	**1.41**×	1202.9 μs	**725.2 μs**	**1.66**×

GPUs (BFV)

Operation	n	$\log_2 q$	GPU with [29] NTT RTX3060Ti	GPU with [29] NTT GTX1080	GPU with new NTT RTX3060Ti	GPU with new NTT GTX1080	[22] Tesla V100	SEAL CPU	T / T_s
...	2^{12}	109	4 μs	4.6 μs	4 μs	4.6 μs	-	14 μs	3.5×
	2^{13}	218	5.1μs	6.2 μs	5.1 μs	6.1 μs	-	58 μs	11.37×

	Instance Size	GPU	GPU Memory (GiB)	vCPUs	Memory (GiB)	Storage (GB)	Network Bandwidth (Gbps)	EBS Bandwidth (Gbps)	On Demand Price/hr*	1-yr ISP Effective Hourly (Linux)	3-yr ISP Effective Hourly (Linux)
Single GPU VMs	g5.xlarge	1	24	4	16	1x250	Up to 10	Up to 3.5	$1.006	$0.604	$0.402
	g5.2xlarge	1	24	8	32	1x450	Up to 10	Up to 3.5	$1.212	$0.727	$0.485
	g5.4xlarge	1	24	16	64	1x600	Up to 25	8	$1.624	$0.974	$0.650
	g5.8xlarge	1	24	32	128	1x900	25	16	$2.448	$1.469	$0.979
	g5.16xlarge	1	24	64	256	1x1900	25	16	$4.096	$2.458	$1.638
Multi GPU VMs	g5.12xlarge	4	96	48	192	1x3800	40	16	$5.672	$3.403	$2.269
	g5.24xlarge	4	96	96	384	1x3800	50	19	$8.144	$4.886	$3.258
	g5.48xlarge	8	192	192	768	2x3800	100	19	$16.288	$9.773	$6.515

	500	64.16 W	18786647 μs	48.60 W	337236 μs	
2^{15}	1	59.55 W	186334 μs	44.29 W	3382 μs	
	10	59.46 W	1796976 μs	45.48 W	36301 μs	
	100	60.89 W	17822724 μs	55.71 W	338212 μs	
	500	66.93 W	89749372 μs	65.13 W	342199 μs	

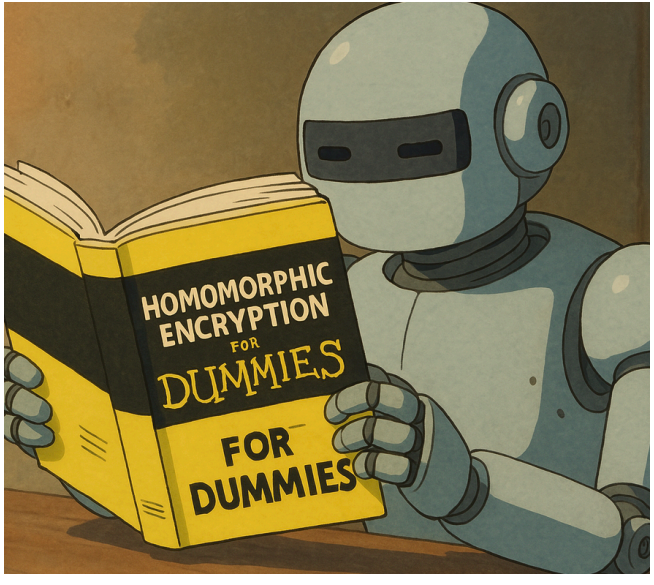
TX 3060Ti)

nverse NTT

			T.W.	T
		s	**15.4 μs**	**1.03×**
		s	19.4 μs	0.91×
		s	**26.6 μs**	**1.10×**
		s	**50.1 μs**	**1.05×**
		s	**91.1 μs**	**1.06×**
		s	**20.5 μs**	**1.03×**
		s	**28.6 μs**	**1.21×**
		s	**49.9 μs**	**1.19×**
		μs	**93.9 μs**	**1.29×**
		μs	**173.6 μs**	**1.29×**
		s	**24.1 μs**	**1.01×**
		s	**53.2 μs**	**1.23×**

2^{14}	32	57.3 μs	**50.1 μs**	**1.14×**	64.5 μs	**51.8 μs**	**1.24×**	121.8 μs	**84.6 μs**	**1.44×**	143.3 μs	**97.2 μs**	**1.47×**
	64	112.6 μs	**96 μs**	**1.17×**	147.4 μs	**99.3 μs**	**1.48×**	218.5 μs	**160.5 μs**	**1.36×**	266.8 μs	**180 μs**	**1.48×**
	128	210.7 μs	**176.1 μs**	**1.19×**	277.1 μs	**183 μs**	**1.51×**	420.8 μs	**303.19 μs**	**1.38×**	511.9 μs	**331.7 μs**	**1.54×**
2^{15}	4	25.6 μs	26.4 μs	0.96×	28.7 μs	**25.9 μs**	**1.10×**	35.8 μs	41.2 μs	0.86×	47.3 μs	50.3 μs	0.94×
	16	64.1 μs	**52.2 μs**	**1.22×**	74.7 μs	**53.2 μs**	**1.40×**	147.1 μs	**100 μs**	**1.47×**	170.1 μs	**95.6 μs**	**1.78×**
	32	136.3 μs	**100.3 μs**	**1.35×**	173 μs	**102.4 μs**	**1.69×**	266.2 μs	**191.8 μs**	**1.38×**	325.5 μs	**193.2 μs**	**1.68×**
	64	254.9 μs	**192.1 μs**	**1.32×**	322.2 μs	**193.6 μs**	**1.66×**	514.2 μs	**372.2 μs**	**1.38×**	633.7 μs	**377.7 μs**	**1.67×**
	128	491.1 μs	**362.4 μs**	**1.35×**	623.1 μs	**364.3 μs**	**1.71×**	998.9 μs	**709.3 μs**	**1.41×**	1202.9 μs	**725.2 μs**	**1.66×**

Homomorphic Encryption



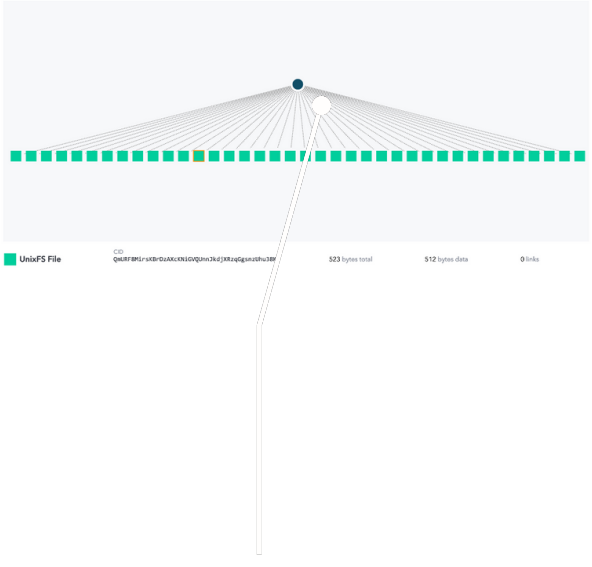
- Homomorphic Encryption: https://asecuritysite.com/openfhe

IPFS

Light-weight Cryptography.
Post Quantum Cryptography
Zero-knowledge Proofs.
Homomorphic Encryption.
> IPFS

Prof Bill Buchanan OBE
http://asecuritysite.com/zero

IPFS



Protobuf UnixFS View on IPFS Gateway

CID QmZqkuqX1qTspb1GgmnzyRFetf1uMyA3CemvvgPZD39sPo
SIZE 33 KB
LINKS 3
DATA

```
▶Object {type: "directory", data: undefined, blockSizes: Array[0]}
```

	PATH	CID
0	1004 - Batman - alt.txt	QmY2JkRD74B1Sb8NRN8EpC1117h6VsZEX7oV16HH78aT11
1	1004 - Batman - transcript.txt	QmQM43UESfCSSQfFAheJEq5jduc2fgx1HVX1Mi3zWC66uc
2	1004 - Batman.png	QmbBnZBudfuGbbzDRTAhSVEYsoxCAe5yBimuuYvYu3BMHG

Protobuf UnixFS View on IPFS Gateway

CID QmY2JkRD74B1Sb8NRN8EpC1117h6VsZEX7oV16HH78aT11
SIZE 185 B
LINKS 0
DATA

```
▼Object {type: "file", data: Buffer[174], blockSizes: Array[0]}
  type: "file"
▼ data: Buffer[174]
    0: 73
    1: 39
    2: 109
    3: 32
    4: 114
    5: 101
    6: 97
    7: 108
    8: 108
    9: 121
    10: 32
```

CID INFO

QmZqkuqX1qTspb1GgmnzyRFetf1uMyA3CemvvgPZD39sPo

base58btc - cidv0 - dag-pb - sha2-256-256-aae5699dbbf2a3...

BASE - VERSION - CODEC - MULTIHASH

MULTIHASH

0x1220aae5699dbbf2a37aa4da529084e2cd82ebacd63c6468becce651522616878112

HASH DIGEST

0x12 = sha2-256
0x20 = 256 bits

Next Generation Crypto

Light-weight Cryptography.
Post Quantum Cryptography
Zero-knowledge Proofs.
Homomorphic Encryption.
zkSnarks, Range-proofs
IPFS

Prof Bill Buchanan OBE
http://asecuritysite.com/encryption