Tomorrow's Cyber Security, Today
IRONCAP

Post-Quantum Cyber Security

Tomorrow's Cyber Security, Today

IronCAP is a collection of post-quantum cryptographic (PQC) technologies, comprising of NIST approved technologies as well as our patented technologies. IronCAP enables today's applications smooth upgrade to become quantum-safe with ease. IronCAP helps you protect against future attacks from the world of quantum computers.

01 COMMUNIQUE

September 2023

Disclaimer

This White Paper may contain forward-looking statements, including statements regarding the cryptographic technologies (the "Technology") invented by 01 Communique Laboratory Inc. (the "Company"). These forward-looking statements involve known and unknown risks, uncertainties and other factors which may cause the actual results, performance or achievements of the Technology to be materially different from any future results, performance or achievements expressed or implied by such forward-looking statements.

A number of factors could cause actual results to differ materially from those in the forward-looking statements, including, but not limited to, rapid changing in the field of computer hardware and software, competition, changes in technology and government policies. In light of the significant uncertainties inherent in the forward-looking statements included herein, the inclusion of such information should not be regarded as a representation by the Company as facts.

The Company believes that the expectations reflected in these forward-looking statements are reasonable; however, no assurance can be given that these expectations will prove to be correct and such forward-looking statements included in this presentation should not be relied upon. In addition, these forward-looking statements relate to the date on which they are made. The Company disclaims any intention or obligation to update or revise any forward-looking statements, whether as a result of new information, future events or otherwise.

This White Paper is for information purposes only. the Company does not guarantee the accuracy of the conclusions reached in this paper, and the White Paper is provided "as is" with no representations and warranties, express or implied, whatsoever. All warranties are expressly disclaimed. The Company expressly disclaims all liability for and damages of any kind arising out of the use, reference to, or reliance on any information contained in this White Paper, even if advised of the possibility of such damages. In no event will the Company be liable to any person or entity for any direct, indirect, special or consequential damages for the use of, reference to, or reliance on this White Paper or any of the content contained herein.

Recipients are specifically notified as follows:

No offer of securities: This White Paper does not constitute a prospectus nor offer document of any sort and is not intended to constitute an offer or solicitation of securities or any other investment or other product in any jurisdiction.

No representations: No representations or warranties have been made to the recipient or its advisers as to the accuracy or completeness of the information, statements, opinions or matters (express or implied) arising out of, contained in or derived from this White Paper or any omission from this document or of any other written or oral information or opinions provided now or in the future to any interested party or their advisers. No representation or warranty is given as to the achievement or reasonableness of any plans, future projections or prospects and nothing in this document is or should be relied upon as a promise or representation as to the future. To the fullest extent, all liability for any loss or damage of whatsoever kind (whether foreseeable or not) which may arise from any person acting on any information and opinions contained in this White Paper or any information which is made available in connection with any further enquiries, notwithstanding any negligence, default or lack of care, is disclaimed.

Table of Contents

EXECUTIVE SUMMARY

Today, technology has truly become part of our lives. We use the Internet, we all share in the cloud, and we create and store data. It is hard to imagine what the world would be like without technology. The more we rely on technology, the more imperative it is to protect ourselves from cyber-attacks.

To many, cybercrime is a distant event which appears on headline news affecting only the well-known names. However, over the past few years, things have changed dramatically. Recent statistics remind us that cybercrimes are closer than we think: with nearly 70 percent of small to medium sized businesses having experienced some form of cybersecurity attack and just under 60 percent having suffered from a data breach of some form in the last 12 months[1]. This is in addition to cyberattacks on individuals which have largely gone unreported. These statistics and experience reflect the challenge we face as we live through a period of unparalleled digital change embracing mobile, Internet of Things, Artificial Intelligence and cloud computing which together result in multi-faceted cyber-attack opportunities. These risks are likely going to increase in the future as quantum computing becomes more accessible. With a quantum computer's extraordinary computation power, what would take a conventional computer over 100 years to decode, will only take seconds, rendering most encryption obsolete and yet quantum computing is no longer fictitious. Are we prepared for its arrival? Should we sit back and rely on current cyber technology to protect us?

IronCAP™ Cryptography (ICC) is a collection of post-quantum cryptographic (PQC) technologies, comprising of technologies selected for standardization by NIST, as well as our patented technologies (US Patent No. 11,271,715). IronCAP is closely following the PQC standardization development at the Computer Security Resource Center (CSRC) of the National Institute of Standards and Technology (NIST). IronCAP includes the PQC technologies NIST selected for standardization from the third-round standardization process:

Digital signature mechanisms:

- CRYSTALS-Dilithium
- Falcon
- SPHINCS+

Key encapsulation mechanisms (KEM):

- CRYSTALS-Kyber
- Classic McEliece[2]

Additional PQC mechanisms will be added to IronCAP to reflect the NIST standardization development.

In addition, IronCAP supports Modern McEliece, our patented invention (US Patent No. 11,271,715). Staying at the leading edge of PQC, IronCAP's Modern McEliece is a forward-thinking key encapsulation mechanism developed in parallel to NIST's evaluation of code-based technology throughout the selection process. Modern McEleice is an enhanced

[1] Data obtained from "The 2018 State of Cybersecurity in Small & Medium Size Businesses study" conducted by Keeper Security
[2] Advancing to NIST 4th round selection process.

version of Classic McEliece with optimization of key size and performance offering improved security over Classic McEliece. With anticipation of Classic McEliece selection for standardization by NIST, IronCAP is staying even a step ahead. Modern McEliece provides an industry unique alternative to NIST technologies.

IronCAP enables smooth adoption by any devices today without re-inventing their security principles while neutralizing the profound threat of quantum computers on cybersecurity with our suite of industry recognized PQC technologies.

Like most adversaries, quantum attacks will come when we are most unprepared. Armed with IronCAP on conventional computers today will safeguard data now while guarding against any unexpected quantum attacks in the future.

1. INTRODUCTION

The purpose of this White Paper is to provide an overview of IronCAP by outlining its design concept, technical framework and the key product offerings.

IronCAP™ is a visionary **C**yber **A**ttack **P**rotection ("CAP") system which aims to guard against future quantum computing threats while being readily used in today's conventional computing world.

The key advantages of IronCAP™ are:

Standard Compliant

Protection with state-of-the-art cryptographic technologies standardized through the rigorous selection process by NIST. Readily integrate through standard interfaces such as PKCS#11 and OpenPGP (RFC4880).

Quantum-safe

Guard against future attacks from quantum computers.

Practical

Applicable in today's conventional devices while safe against future quantum computers.

Efficient

Quick and efficient encryption/decryption/key-generation processes. Our implementation takes advantage of the Advanced Vector Extensions 2 (AVX2) capability of most modern CPU to ensure the highest possible throughput.

Credible

Elevate from today's proven best-in-class code-base cryptography theory and NIST selected mechanisms.

Versatile

Broadly applicable to a cross-platform of vertical applications such as, but not limited to, Email encryption, File encryption, Digital signature, Blockchain security, Remote access / VPN, Cloud storage, Credit card security, General Internet communication security, etc. Available with OpenSSL support.

2. BACKGROUND

There are many types of malicious activities[3], over time, security tactics have been developed to counter malicious attacks and a key aspect is to prevent unauthorized comprehension of data. This is critical in addressing the ever-evolving illegitimate and malicious means of gaining access to data. Many technologies have been developed to make the Internet safe against such mischievous activities. The fundamental approach is to implement encryption to ensure that no one can understand the content of the communication session even if security has been breached.

Encryption Techniques and Security Strength - Brute Force Time

A common type of encryption technique is the asymmetrical encryption method using a pair of private and public keys. This technique has been proven to work effectively over the past decades. Examples of asymmetrical encryptions include RSA (Rivest-Shamir-Adleman) and El Gamal. This type of encryption uses one of the keys to encrypt data so that only the entity having the other key can decrypt it.

In general, each pair of public and private keys is generated whereby the private key represents two large random prime numbers; and the public key is the product (i.e. the multiple) of these two large prime numbers. Security of asymmetrical encryption lies in the difficulty of factoring the two prime numbers from the product i.e. figure out the private key from the public key. The larger the prime numbers, the longer it takes to factor (decode) and the time required to factor is often regarded as "brute force time".

A key size of 4096 bits in traditional cryptography is believed to be unbreakable in an acceptable time frame using brute force factoring. However, this time period will only be reduced as technology and computer processing capability improves.

[3] Malicious activity includes inserting spyware into an end-user computer to surreptitiously copy Internet activity such as User Name and Password transmission for various sensitive accounts. Anti-virus programs are one tool used to combat this type of attack. Another malicious activity is spoofing a web site, such as a banks' online login page, to look the same as the original. When a user enters login credentials, this forged site captures the sensitive login credentials. Digital Certificates are a tool used to secure the Internet against this type of attack. Digital Certificates help to guarantee that the site you are visiting is really the site operated by the original intended organization instead of a "look-a-like" spoofed site operated by a malicious party. Digital Certificates also allow software publishers to digitally sign executable files to prove legitimacy. Even though there are many ways to block malicious activities, there is an ever-present risk of becoming a victim of a cyber-attack. For example, malicious attackers are constantly working on ways to bypass anti-virus software. Malicious parties may also be able to gain access to the database of a public server, such as a bank or a social media site, by-passing the login process. Malicious parties may also be able to tap into a communication session between an end user and a website they are accessing and collect data as a "man in the middle".

3. THE ARRIVAL OF THE QUANTUM ERA

Quantum Computing is a New Cybersecurity Threat

A new breed of computer has been in development since the early 1980s referred to as quantum computers. Quantum computing theory was first introduced as a concept by Richard Feynman. It has been researched extensively and is considered the destructor of the present modern asymmetric cryptography. Quantum computers adopt unique principles, namely superposition[4] and entanglement. These principles enable quantum computers to factor large numbers in polynomial time to break the public/private key mechanism in a much shorter time than conventional computers. The application of quantum computing to cryptography has become one of the most promising possibilities, and has attracted a lot of scientific attention, research[5] and resources. Various simulations and experiments have been conducted to compress the brute force time. With such focus, it is not surprising to see that recent development on period finding mechanism[6] is making quantum application to cryptography more of reality than theoretical application.

According to the U.S. National Institute of Standard and Technology (NIST[7]), "regardless of whether we can estimate the exact time of the arrival of the quantum computing era, we must begin now to prepare our information security systems to be able to resist quantum computing" It also recognizes that a large international community has emerged to address the issue of information security in a quantum computing future. Efforts to develop quantum-resistant technologies is intensifying reflecting the urgency and determination for the cybersecurity to win in this quantum race.

[4] Computers work with information in the form of bits as a "1" or a "0" at any one time. If we send a question to a computer it has to proceed with orderly, linear fashion to find the answer. Quantum computers adopt superposition rules, its bits can be 1 or 0, or 0 and 1 at the same time. In this superposed state, a quantum bit exists as two equally probably possibilities at the same time so when a single quantum bit can be in two states at the same time, it can perform two calculations at the same time. Two quantum bits could perform four simultaneous calculations, three quantum bits could perform eight; and so on… creating exponentially increased calculation power.

[5] Vasileios Mavroeidis, Kamer Vishi, Mateusz D. Zych, Audun Jøsang, "The Impact of Quantum Computing on Present Cryptography", March 2018. https://arxiv.org/pdf/1804.00200.pdf

[6] Shor's Algorithm, which is designed to run on a Quantum computer, is the process of period-finding which is done using Quantum Fourier Transform (QFT). The QFT can be used to determine the period of a function f(x). QFT processing can be done efficiently on a quantum computer because all of the experiments can be run at once in superposition, with bad experiments deteriorating from destructive interference effects and the good experiments dominating from constructive interference effects. Once the period-finding mechanism of the QFT becomes available, it can be exploited to find patterns in the mathematical structure of the number being factored. While not yet a commodity item, quantum computers will be at least available via the cloud in the foreseeable future.

[7] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, Daniel Smith-Tone, "Report on Post-Quantum Cryptography made by U.S. National Institute of Standard and Technology", April 2016. https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf

4. OUR OFFERINGS

Tomorrow's Cyber Security, Today

IRONCAP

General

IronCAP™ Cryptography (ICC) is a collection of post-quantum cryptographic (PQC) technologies, available as a cryptographic library. IronCAP comprises of PQC technologies selected for standardization by NIST, as well as our patented invention for encryption:

Digital signature mechanisms:

- CRYSTALS-Dilithium
- Falcon
- SPHINCS+

Key encapsulation mechanisms (KEM):

- CRYSTALS-Kyber
- Classic McEliece
- Modern McEliece

CRYSTALS-Dilithium, Falcon, SPHINCS+, CRYSTALS-Kyber have been selected for standardization by NIST at Round 3 of the selection process. Classic McEliece is the code-based candidate for KEM in the ongoing Round 4 consideration.

CRYTSALS-Dilithium, or Dilthium, is a lattice-based digital signature algorithm based on Fiat-Shamir paradigm. IronCAP supports the Dilithium and Diltihium-AES variants.

FALCON (Fast Fourier Latticed-based Compact Signatures over NTRU) is a lattice-based signature scheme utilizing the "hash-and-sign" paradigm.

SPHINCS+ is a stateless hash-based signature scheme. IronCAP supports the SPHINCS+ Robust and SPHINCS+ Simple variants.

CRYSTALS-Kyber, or Kyber, is a CCA (Chosen Ciphertext Attack) secure lattice-based key encapsulation mechanism. IronCAP supports the Kyber and Kyber-90S variants.

Classic McEliece is a code-based key encapsulation mechanism that uses a binary Goppa code in the Niederreiter variant of the McEliece cryptosystem combined with standard techniques to achieve CCA security.

Modern McEliece is our patented invention (US Patent No. 11,271,715), being the forward-thinking enhancement to Classic McEliece which

- eliminates M&N system's particular shortcoming of large key size by increasing the encryption/decryption efficiency
- maintains M&N system's key advantages: secure and fast with very low complexity
- further enhances the security level of the M&N system.

IronCAP™ achieves a remarkable computational efficiency and a reduction of the public key size as compared to convention M&N system. The technical insight is outlined in the Appendix.

IronCAP™ can be applied to many vertical applications. Among them, the most notable are:

- Email encryption
- File encryption
- Digital signature
- Blockchain security
- Remote access / VPN
- Cloud storage
- Credit card security
- General Internet communication security

Vendors of the above applications can easily adopt IronCAP™ to transform or to up-scale their cyber security solutions to a post-quantum level readiness for today as well as the world of quantum computing.

In our immediate product roadmap, we are planning to offer an email/file encryption and digital signing product called IronCAP™ SaaS that tightly integrates with popular email clients such as Outlook and webmail like Gmail, etc. IronCAP SaaS is an end-to-end encryption that directly delivers emails to the recipients in an encrypted form secured by IronCAP™ post-quantum cryptographic technologies. This unique feature aligns fully with the associated email applications allowing users to read secured messages from the email application. We believe this is the only way to deliver a seamless user experience and a stepped improvement from many secure email services that store their users' messages and require recipients to read the message from their sites. This eliminates the server silo as a central point of target for cyber-attacks.

Compliance

NIST

IronCAP consists of a collection of post-quantum cryptographic mechanisms. Besides our forward-thinking Modern McEliece KEM, IronCAP also supports PQC mechanisms NIST recommended for standardization: CRYSTALS-Kyber, Classic McEliece, CRYSTALS-Dilithium, Falcon, SPHINCS+. Following the NIST standardization process closely, other NIST endorsed cryptographies are continuously being added.

PKCS#11

IronCAP provides a PKCS#11 standard interface for seamless application integration through ICCHSM, an IronCAP toolkit. IronCAP's ICCHSM allows applications to use IronCAP cryptographic mechanisms transparently, without being aware of implementation and

without the need for any additional hardware modules. The IronCAP's ICCHSM acts as the application's PKCS#11 provider. IronCAP cryptographic keys are available as vendor specific key types in PKCS#11 for application integration, until IronCAP cryptographic key types are officially registered with an RFC. A PKCS#11-compliant hardware security module (HSM) based on IronCAP Key HSM firmware upgrade is in the roadmap.

OpenSSL

IronCAP's ICC OpenSSL is another IronCAP toolkit. ICC OpenSSL is an OpenSSL extension supporting IronCAP cryptographic mechanisms. OpenSSL is used extensively in the industry and as an integral part of the overall security workflow. With IronCAP's ICC OpenSSL, vendors can quickly introduce post-quantum technologies into their security offering with minimal changes to their workflow. Crucial cryptographic functions such as key generation, data encryption and decryption, data signature and signature verification, and X509 generation and verification using IronCAP cryptographic mechanisms can be done natively in OpenSSL using OpenSSL commands.

OpenPGP (RFC4880)

IronCAP allows a seamless integration of post-quantum safety support into software solutions already familiar to their users by adding a new public key cipher into Libgcrypt (the back-end cryptographic library of GnuPG and a number of other products). To ensure backward compatibility during the transition time, this can be used together with the conventional public key ciphers, like RSA, ECC, etc. To further ensure a smooth transition, the users can use their existing GnuPG RSA, ECC, etc. keys with IronCAP while provided with an option of migrating to the post-quantum crypto technology.

IronCAP Crypto will also be delivered bundled with the OpenPGP Card - compliant IronCAP Key hardware security module (HSM). This provides a way of using IronCAP Crypto private keys seamlessly and without any exposure to the operating system apps, e.g. protected from any malicious activity and spyware.

Appendix A – Key and Signature Sizes of different PQC Mechanisms

The key and signature sizes are shown in bytes for different bit security levels are shown below, as compared to RSA with 4096 bits.

128 Bit Security Level

Mechanism	Public Key Size	Private Key Size	Signature Size
Modern McEliece with SHAKE 256 hash	304,744	471,882	N/A
Modern McEliece with SHA 256 hash	304,744	471,947	N/A
Kyber 90S with SHA 256 hash	886	1,718	N/A
Kyber with SHAKE 256 hash	886	1,718	N/A
Classic McEliece with SHAKE256 hash	261,208	6578	N/A
Sphincs+ Robust with SHA256 hash	114	147	17,088
Sphincs+ Simple with SHA256 hash	114	147	17,088
Sphincs+ Robust with SHAKE256 hash	114	147	17,088
Sphincs+ Simple with SHAKE256 hash	114	147	17,088
Sphincs+ Robust with Haraka hash	114	147	17,088
Sphincs+ Simple with Haraka hash	114	147	17,088
Dilithium with SHAKE256 hash	1,398	2,614	2,420
Dilithium AES with SHAKE256 hash	1,398	2,614	2,420
FALCON with SHAKE256 hash	983	1,367	690
RSA with 4096 bits	550	2348	512

192 Bit Security Level

Mechanism	Public Key Size	Private Key Size	Signature Size
Modern McEliece with SHAKE 256 hash	494,696	908.804	N/A
Modern McEliece with SHA 256 hash	494,696	908,820	N/A
Kyber 90S with SHA 256 hash	1,270	2,486	N/A
Kyber with SHAKE 256 hash	1,270	2,486	N/A
Classic McEliece with SHAKE256 hash	524,248	13,694	N/A
Sphincs+ Robust with SHA256 hash	131	179	35,664
Sphincs+ Simple with SHA256 hash	131	179	35,664
Sphincs+ Robust with SHAKE256 hash	131	179	35,664
Sphincs+ Simple with SHAKE256 hash	131	179	35,664
Sphincs+ Robust with Haraka hash	131	179	35,664
Sphincs+ Simple with Haraka hash	131	179	35,664
Dilithium with SHAKE256 hash	2,038	4,086	3,293
Dilithium AES with SHAKE256 hash	2,038	4,086	3,293
FALCON[8] with SHAKE256 hash	1,879	2,391	1,330
RSA with 4096 bits	550	2348	512

[8] FALCON with 192 bit security level is undefined. For consistency with other PQC mechanisms, FALCON 192 bit security is implemented as FALCON 256 bit security (FALCON 1024).

256 Bit Security Level

Mechanism	Public Key Size	Private Key Size	Signature Size
Modern McEliece with SHAKE 256 hash	1,281,384	2,061,204	N/A
Modern McEliece with SHA 256 hash	1,281,384	2,061,092	N/A
Kyber 90S with SHA 256 hash	1,654	3,254	N/A
Kyber with SHAKE 256 hash	1,654	3,254	N/A
Classic McEliece with SHAKE256 hash	1,357,912	14,206	N/A
Sphincs+ Robust with SHA256 hash	147	212	49,856
Sphincs+ Simple with SHA256 hash	147	212	49,856
Sphincs+ Robust with SHAKE256 hash	147	212	49,856
Sphincs+ Simple with SHAKE256 hash	147	212	49,856
Sphincs+ Robust with Haraka hash	147	212	49,856
Sphincs+ Simple with Haraka hash	147	212	49,856
Dilithium with SHAKE256 hash	2,678	4,950	4,595
Dilithium AES with SHAKE256 hash	2,678	4,950	4,595
FALCON with SHAKE256 hash	1,879	2,391	1,330
RSA with 4096 bits	550	2348	512

Appendix B – Performance Benchmarking

Scenario 1: PQC vs RSA in Intel Xeon Bronze 3104 CPU @1.70 GHz, with 8GB RAM

Performance benchmarking were performed for IronCAP PQC mechanisms at 128 bit security as compared to RSA with 4096 bits as the benchmark. The following table shows the duration it takes to perform key generation, signing/verification and encryption/decryption against a random 32 bytes of binary data. The results are reported in seconds.

Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	0.289922	0.00025146	0.005966856		
Modern McEliece with SHA 256 hash	0.2983278	0.000251547	0.005973249		
Kyber 90S with SHA 256 hash	0.000152688	0.000150967	2.46628E-05		
Kyber with SHAKE 256 hash	9.54616E-05	9.36264E-05	2.51964E-05		
Classic McEliece with SHAKE256 hash	0.9145761	0.000227824	0.06903164		
Sphincs+ Robust with SHA256 hash	0.006303783			0.1451732	0.004556486
Sphincs+ Simple with SHA256 hash	0.003138995			0.0731765	0.002190668
Sphincs+ Robust with SHAKE256 hash	0.01090098			0.2528755	0.007808527
Sphincs+ Simple with SHAKE256 hash	0.005625278			0.131414	0.00391048
Sphincs+ Robust with Haraka hash	0.01006464			0.2399856	0.007837058
Sphincs+ Simple with Haraka hash	0.005525761			0.1330868	0.004212015
Dilithium with SHAKE256 hash	0.000228186			0.000955948	0.000116495
Dilithium AES with SHAKE256 hash	0.000464569			0.00117918	0.000218207
FALCON with SHAKE256 hash	0.04693281			0.010254	3.90914E-05
RSA	1.09731	1.7286E-04	8.4712E-03	8.1312E-03	1.5318E-04

The following table shows the percentage improvement ICC PQC over RSA by using

$$(RSA\ performance - PQC\ mechanism\ performance)\ /\ RSA\ performance\ x\ 100\%.$$

A negative value indicates the PQC mechanism is slower.

PQC Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	74%	-45%	30%		
Modern McEliece with SHA 256 hash	73%	-46%	29%		
Kyber 90S with SHA 256 hash	100%	13%	100%		
Kyber with SHAKE 256 hash	100%	46%	100%		
Classic McEliece with SHAKE256 hash	17%	-32%	-715%		
Sphincs+ Robust with SHA256 hash	99%			-1685%	-2875%
Sphincs+ Simple with SHA256 hash	100%			-800%	-1330%
Sphincs+ Robust with SHAKE256 hash	99%			-3010%	-4998%
Sphincs+ Simple with SHAKE256 hash	99%			-1516%	-2453%
Sphincs+ Robust with Haraka hash	99%			-2851%	-5016%
Sphincs+ Simple with Haraka hash	99%			-1537%	-2650%
Dilithium with SHAKE256 hash	100%			88%	24%
Dilithium AES with SHAKE256 hash	100%			85%	-42%
FALCON with SHAKE256 hash	96%			-26%	74%

Scenario 2: PQC vs RSA in Intel Pentium Dual-Core CPU E5500 @2.80GHz with 8GB RAM

The same comparison was performed using a platform with a lower performance. The following table shows the duration it takes to perform key generation, signing/verification and encryption/decryption against a random 32 bytes of binary data. The results are reported in seconds.

Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	0.3141911	0.000223283	0.005426704		
Modern McEliece with SHA 256 hash	0.3009636	0.000222983	0.00544575		
Kyber 90S with SHA 256 hash	0.000138673	0.000143399	2.97073E-05		
Kyber with SHAKE 256 hash	0.000093806	0.00010174	2.97335E-05		
Classic McEliece with SHAKE256 hash	0.5135994	0.000144695	0.05198889		
Sphincs+ Robust with SHA256 hash	0.004689347			0.108597	0.003414428
Sphincs+ Simple with SHA256 hash	0.002339975			0.05449549	0.001629677
Sphincs+ Robust with SHAKE256 hash	0.008630769			0.1998344	0.006164597
Sphincs+ Simple with SHAKE256 hash	0.004456641			0.1038335	0.003086016
Sphincs+ Robust with Haraka hash	0.007366009			0.1751756	0.005702856
Sphincs+ Simple with Haraka hash	0.004093561			0.09826776	0.003109643
Dilithium with SHAKE256 hash	0.000211244			0.000923099	0.000108694
Dilithium AES with SHAKE256 hash	0.000392198			0.001167424	0.000187855
FALCON with SHAKE256 hash	0.02929586			0.009922308	3.79234E-05
RSA	1.711391	3.1729E-04	1.7746E-02	1.7246E-02	2.9477E-04

The following table shows the percentage improvement PQC over RSA.

PQC Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	82%	30%	69%		
Modern McEliece with SHA 256 hash	82%	30%	69%		
Kyber 90S with SHA 256 hash	100%	55%	100%		
Kyber with SHAKE 256 hash	100%	68%	100%		
Classic McEliece with SHAKE256 hash	70%	54%	-193%		
Sphincs+ Robust with SHA256 hash				-530%	-1058%
Sphincs+ Simple with SHA256 hash				-216%	-453%
Sphincs+ Robust with SHAKE256 hash				-1059%	-1991%
Sphincs+ Simple with SHAKE256 hash				-502%	-947%
Sphincs+ Robust with Haraka hash				-916%	-1835%
Sphincs+ Simple with Haraka hash				-470%	-955%
Dilithium with SHAKE256 hash				95%	63%
Dilithium AES with SHAKE256 hash				93%	36%
FALCON with SHAKE256 hash				42%	87%

Scenario 3: PQC vs RSA in Raspberry Pi 4 Model B with 4GB RAM

The same comparison was performed on a Raspberry Pi. The following table shows the duration it takes to perform key generation, signing/verification and encryption/decryption against a random 32 bytes of binary data. The results are reported in seconds.

Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	0.3027099	0.00021961	0.007731619		
Modern McEliece with SHA 256 hash	0.348769	0.000219008	0.007690118		
Kyber 90S with SHA 256 hash	0.000172756	0.00016711	2.67669E-05		
Kyber with SHAKE 256 hash	0.000100413	9.93491E-05	2.67806E-05		
Classic McEliece with SHAKE256 hash	0.3978583	0.000214297	0.03900254		
Sphincs+ Robust with SHA256 hash	0.004786259			0.1104284	0.003481486
Sphincs+ Simple with SHA256 hash	0.002382461			0.05515846	0.001657092
Sphincs+ Robust with SHAKE256 hash	0.009879691			0.2286099	0.007082461
Sphincs+ Simple with SHAKE256 hash	0.005072502			0.1182259	0.003505584
Sphincs+ Robust with Haraka hash	0.0116702			0.2786024	0.009123787
Sphincs+ Simple with Haraka hash	0.006481329			0.1562417	0.004945097
Dilithium with SHAKE256 hash	0.000230773			0.000969098	0.00011534
Dilithium AES with SHAKE256 hash	0.000525145			0.001440861	0.000243052
FALCON with SHAKE256 hash	0.04089369			0.01123561	4.72137E-05
RSA	4.178938	8.8520E-04	4.5228E-02	4.3442E-02	7.7142E-04

The following table shows the percentage improvement PQC over RSA.

PQC Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	93%	75%	83%		
Modern McEliece with SHA 256 hash	92%	75%	83%		
Kyber 90S with SHA 256 hash	100%	81%	100%		
Kyber with SHAKE 256 hash	100%	89%	100%		
Classic McEliece with SHAKE256 hash	90%	76%	14%		
Sphincs+ Robust with SHA256 hash	100%			-154%	-351%
Sphincs+ Simple with SHA256 hash	100%			-27%	-115%
Sphincs+ Robust with SHAKE256 hash	100%			-426%	-818%
Sphincs+ Simple with SHAKE256 hash	100%			-172%	-354%
Sphincs+ Robust with Haraka hash	100%			-541%	-1083%
Sphincs+ Simple with Haraka hash	100%			-260%	-541%
Dilithium with SHAKE256 hash	100%			98%	85%
Dilithium AES with SHAKE256 hash	100%			97%	68%
FALCON with SHAKE256 hash	99%			74%	94%

Scenario 4: PQC improvement with AVX2 in Intel Core i7-4790K CPU @4.00GHz with 16GB RAM

The following table shows the duration it takes to perform key generation, signing/verification and encryption/decryption against a random 32 bytes of binary data, without AVX2 optimization. The results are reported in seconds.

PQC Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	0.1323434	0.000106919	0.002656011		
Modern McEliece with SHA 256 hash	0.1097531	0.00010592	0.00267218		
Kyber 90S with SHA 256 hash	7.50724E-05	6.80927E-05	1.16478E-05		
Kyber with SHAKE 256 hash	4.39322E-05	4.23123E-05	1.08756E-05		
Classic McEliece with SHAKE256 hash	0.1370745	8.08042E-05	0.01466486		
Sphincs+ Robust with SHA256 hash	0.002927089			0.06709148	0.002118651
Sphincs+ Simple with SHA256 hash	0.001718804			0.0337845	0.001011411
Sphincs+ Robust with SHAKE256 hash	0.005720812			0.1332545	0.004121264
Sphincs+ Simple with SHAKE256 hash	0.002973473			0.06914927	0.002054511
Sphincs+ Robust with Haraka hash	0.004624258			0.1108451	0.003655623
Sphincs+ Simple with Haraka hash	0.002734724			0.06560144	0.002082376
Dilithium with SHAKE256 hash	0.000106014			0.000352272	5.29204E-05
Dilithium AES with SHAKE256 hash	0.000213574			0.000573335	9.75537E-05
FALCON with SHAKE256 hash	0.01185782			0.004157309	1.6113E-05

The following table shows the duration it takes to perform key generation, signing/verification and encryption/decryption against a random 32 bytes of binary data, with AVX2 optimization. The results are reported in seconds.

PQC Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	0.141451	3.92473E-05	0.002601264		
Modern McEliece with SHA 256 hash	0.1202533	4.01579E-05	0.002614462		
Kyber 90S with SHA 256 hash	1.62492E-05	7.46591E-06	3.9134E-07		
Kyber with SHAKE 256 hash	1.58148E-05	9.38536E-06	3.94542E-07		
Classic McEliece with SHAKE256 hash	0.01463605	3.80929E-05	3.23707E-05		
Sphincs+ Robust with SHA256 hash	0.000593832			0.01378721	0.000622284
Sphincs+ Simple with SHA256 hash	0.000299076			0.006639838	0.000287923
Sphincs+ Robust with SHAKE256 hash	0.000974265			0.02191101	0.000885974
Sphincs+ Simple with SHAKE256 hash	0.000512835			0.01163033	0.000450871
Sphincs+ Robust with Haraka hash	0.00019373			0.004281716	0.000166126
Sphincs+ Simple with Haraka hash	0.000133243			0.003063647	0.000112173
Dilithium with SHAKE256 hash	3.28864E-05			7.20076E-05	0.000013829
Dilithium AES with SHAKE256 hash	2.24222E-05			7.6478E-05	1.03411E-05
FALCON with SHAKE256 hash	0.006588704			0.000212303	1.61821E-05

The following table shows the percentage improvement with AVX2 optimization.

PQC Mechanism	Key Generation	Encryption	Decryption	Signing	Verification
Modern McEliece with SHAKE 256 hash	-7%	63%	2%		
Modern McEliece with SHA 256 hash	-10%	62%	2%		
Kyber 90S with SHA 256 hash	78%	89%	97%		
Kyber with SHAKE 256 hash	64%	78%	96%		
Classic McEliece with SHAKE256 hash	89%	53%	100%		
Sphincs+ Robust with SHA256 hash	80%			79%	71%
Sphincs+ Simple with SHA256 hash	83%			80%	72%
Sphincs+ Robust with SHAKE256 hash	83%			84%	79%
Sphincs+ Simple with SHAKE256 hash	83%			83%	78%
Sphincs+ Robust with Haraka hash	96%			96%	95%
Sphincs+ Simple with Haraka hash	95%			95%	95%
Dilithium with SHAKE256 hash	69%			80%	74%
Dilithium AES with SHAKE256 hash	90%			87%	89%
FALCON with SHAKE256 hash	44%			95%	0%

Appendix C – Modern McEliece

DESIGN CONCEPT

Quantum-Safe Encryption Algorithms

In searching for a quantum-safe cryptography solution, the key requirement is the applicability to the existing digital environment. Looking back at the last digital transformation, the world did not adopt the Internet overnight. Our solution must be deployable to the current environment in preparation for quantum computers which will come to us anytime anywhere! Therefore, the most practical solution is to build on an existing cryptography theory.

A number of theories have been proposed that are theoretically capable of exponentially increasing the brute force time. These are the best candidates to guard against quantum computer attacks. Amongst them, the most notable are:

- Code-based encryption;
- Lattice-based encryption;
- Hash-based encryption;
- Multivariate-based encryption;
- Super-singular isogenies of elliptic curves.

According to the Report on Post-Quantum Cryptography made by U.S. National Institute of Standard and Technology[6] all of them have their pros and cons. Further details are outlined in the next section.

Obviously, it is almost impossible to find a particular theory that is universally supported. Our own review and analysis concluded that the code-based best-in-class McEliece and Niederreiter cryptographic system ("the M&N system") has remained remarkably stable backed by a proven track record to withstand dozens of attack papers over 40 years[9]. Further explanation on the M&N system is provided in the Appendix. In addition, it has tremendous scalability that is capable of providing ample security margin against advances in computer technology, including quantum computers. Its main drawback, however, is the exceptionally large key size rendering it unattractive for some areas of practical usage due to the inefficiency in storage, encryption/decryption time, etc.

Outline of the key cryptography theories

Systems based on multivariate polynomial equations use the difficulty of solving systems of multivariate polynomials over finite fields. Several systems have been proposed over the

[9] Daniel J. Bernstein, Tanja Lange, Christiane Peters, "Attacking and defending the McEliece cryptosystem", 2008. https://cr.yp.to/codes/mceliece-20080722.pdf; Daniel J. Bernstein, Tung Chou, Tanja Lange, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Wen Wang, "Classic McEliece: Conservative Code-Based Cryptography", 2017. https://classic.mceliece.org/nist/mceliece-20171129.pdf; Daniel J. Bernstein, Johannes Buchmann, Erik Dahmen, Post-Quantum Cryptography 2009. https://www.springer.com/gp/book/9783540887010; Nicolas Sendrier, "Code-Based Cryptography: State of the Art and Perspectives", August 2017. https://ieeexplore.ieee.org/document/8012331

past few decades, most of them have been broken. The newest are Simple Matrix Scheme or ABC, presented by Tao et al.[10] and ZHFE, a New Multivariate Public Key Encryption Scheme, presented by Porras et al[11]. The ABC scheme has already been broken as well[12].

Hash-based algorithms can offer relatively small public keys but cannot be used for encryption. It is for signatures only and the number of signatures is limited. It is possible to increase the number of signatures but it would result in very long signature sizes. Another drawback is that a signer needs to keep a track of all of the messages that have been signed.

Key exchange algorithms based on super singular isogenies of elliptic curves seem to be a good post-quantum key exchange candidate but are considerably slower than the other algorithms[13], and like some other proposals, there has not been enough analysis to have much confidence in their security. Another drawback is that the increase in both complexity of theory and lines of code would likely make an implementation of SIDH more expensive to write since it would probably be easier to introduce bugs and harder to find them among the large amount of code[6].

Lattice-based algorithms seem to be simple and effective but use larger keys than the currently used algorithms, and the ring structure of the faster versions might raise some security concerns[11]. Also, it has proven difficult to give precise estimates of the security of lattice schemes against even known cryptanalysis techniques.

These cryptosystems are relatively new and the security estimates for lattice-based cryptosystems have not been finalized yet.

In their work "Post-quantum cryptography - dealing with the fallout of physics success"[14] world renown cryptologists Daniel J. Bernstein and Tanja Lange said: "Much more research is required to gain confidence in the security of lattice-based cryptography. The only systems that have received enough study for us to recommend are the original McEliece/Niederreiter systems".

The McEliece cryptosystem and the Niederreiter cryptosystem are code-based encryption functions which have been mathematically proven to have an exponential relation between key size and the brute force time in the Post-Quantum world. McEliece was first proposed in 1978 and has not been broken since then, it is one of the most successful cryptosystems. According to the Survey on cryptanalysis of code-based cryptography made in May 2018, the scheme has various advantages – firstly, the complexity of encryption and decryption algorithms are equivalent to those of symmetric schemes and thus are very efficient compared to other public key schemes and secondly, the best attacks for solving the SDP are exponential in the code length, which makes code-based schemes of high potential[8].

Although a more secure form of encryption, the McEliece and Niederreiter schemes use a large key size, requiring more storage resources. As for any other class of cryptosystems,

[10] Chengdong Tao, Adama Diene, Shaohua Tang, Jintai Ding, "Simple Matrix Scheme for Encryption", In: Gaborit P. (eds) Post-Quantum Cryptography. PQCrypto 2013. https://link.springer.com/chapter/10.1007/978-3-642-38616-9_16
[11] Jaiberth Porras, John Baena, Jintai Ding, "ZHFE, a New Multivariate Public Key Encryption Scheme", In: Mosca M. (eds) Post-Quantum Cryptography. PQCrypto 2014. https://link.springer.com/chapter/10.1007%2F978-3-319-11659-4_14
[12] Chunsheng Gu, "Cryptanalysis of Simple Matrix Scheme for Encryption", 2016. https://eprint.iacr.org/2016/1075.pdf
[13] Marcus Kindberg, "A Usability Study of Post-Quantum Algorithms", June 2017. https://www.eit.lth.se/sprapport.php?uid=1053
[14] Daniel J. Bernstein and Tanja Lange, "Post-Quantum Cryptography – Dealing with the Fallout of Physics Success", April 2017. https://eprint.iacr.org/2017/314.pdf

the practice of code-based cryptography is a trade-off between efficiency and security. McEliece cryptosystem encryption and decryption processes are fast with very low complexity, but it makes use of large public keys (100 kilobytes to several megabytes).

Technical Explanations

In the classical McEliece and Niederreiter algorithms, the scheme parameters are determined by two key elements m and t and the error correcting code is considered in the classical, i.e., unweighted, Hamming metric. Our variant offers a significant flexibility in choosing the parameter of the code length n based on a third key parameter r and the use of Goppa codes in a weighted Hamming metric. A special type of locator set is used that is a set of rational functions of degree not greater than r where r is greater than 1, and with coefficients from a finite field GF(2^m). This is contrasted with the classical scheme, in which the elements of the field GF(2^m) are used as the locator set. This change in the locator set significantly increases the length of the code, while the calculations remain in the field GF(2^m).

This implementation allows for: 1) the expansion of the selection of a support set, thereby expanding the available private keys; 2) use of rational functions to keep the calculation in a finite field with a comparable code length. For example, for rational functions of degree 2 with coefficients from the field GF(2^m), the code length is $n = 2^{2m-1} + 2^{m-1}$. The practical benefits of using rational functions with different degree are: 1) reducing the amount of CPU cycles needed in the encryption, decryption, and key generation processes; and 2) increasing the security for codes with the same parameters (n, k, d), as in classical Goppa codes.

Using a set of position numerators of degree greater than 1, the degree of Galois field extension m for obtaining a support set L is reduced, thereby reducing the complexity of the calculations in the decoding process. The degree m of the field extension is reduced by r times, where r is the degree of the position numerators.

The cryptosystems we offer provide a public key cryptographic system and method that can be used to build a highly secure system for data storage, access, encryption, decryption, digital signing, digital signature verification, etc.

Technical Insights

The product will now be described with reference to the diagrams (see below) in which like reference numerals refer to like parts throughout. In Fig. 1, a public key cryptographic device 100 is depicted. The cryptographic device 100 receives an encryption key from memory 102, which can be a public key or a private key. The cryptographic device also receives from an input device 104, data that is to be encrypted. The data can include, but is not limited to, a message vector to be securely transmitted from a sender to a receiver, or data from a network interface, a data storage device such as a hard drive, a key board, and the like. As described below, the data to be encrypted can also be the hash value of data to be digitally signed.

The encryption key and encrypted data may be received from inside a computing device, such as a personal computer, from one or more devices within a network or from third party devices outside the network. As described in more detail below, it will be readily understood that the public key cryptographic device can be any device capable of performing the processes described herein whether integrated into a single semiconductor package or distributed amongst several semiconductor devices contained within a single computer or server or distributed over multiple devices within one or more networks.

The cryptographic device 100 includes an input/output device 106, which can, for example, be a network communication interface, for receiving the plain data from the input device 104 and receiving the encryption key. The plain data and encryption key are then forwarded to an Input/Output Bridge 108 and a Memory Bridge 110 for storage in system memory 112. In exemplary embodiments the System Memory 112 may contain operating instructions such as, but not limited to, the Operating System 114. In addition to the operating system as well as other operating instructions 114 that are stored in system memory 112, the system memory includes the processing instructions of a cryptographic engine 116. The cryptographic engine 116 provides the operational instructions for the cryptographic functions such as encryption, decryption, digital signature, verification of digital signature, etc.

The cryptographic processing of the encrypted data is performed in the CPU 118 that is linked to system memory 112 via a Memory Bus. The CPU 118 can be implemented as a parallel co-processor, a field programmable gate array (FPGA), microprocessor, or the like, as is well understood.

Where all components of the system are contained within a single device, as depicted in Fig. 1, the cryptographic device 100 can be implemented as a single purpose computing device, e.g., a special device performing one or more special cryptography functions like a secure key device, a credit card chip, passport chip, etc.) Alternatively, the components and functioning depicted in Fig.1 can be distributed within a multiple purpose computing device, e.g., a general computer or server, or distributed over multiple devices within a network. For example, the functioning can be implemented on a cluster of server computers in a manner that is well-known.

The embodiment of Fig. 1 can be implemented on a computer network such as the Internet that is strengthened against cyberattack from both classical computers and quantum computers and has a manageable key size and improved computational efficiency using a

variant of the McEliece and Niederreiter schemes. In the classical McEliece and Niederreiter algorithms, the scheme parameters are determined by two key elements m and t and the error correcting code is considered in the classical, i.e., un-weighted, Hamming metric. Our variant provides significant flexibility in choosing the parameter of the code length n based on a third key parameter r and the use of Goppa codes in a weighted Hamming metric. In our algorithm, a special type of locator set, L^*, is used that is a set of rational functions of degree not greater than r where r is greater than 1, and with coefficients from a finite field $GF(2^m)$. This is contrasted with the classical scheme, in which the elements of the field $GF(2^m)$ are used as the locator set. This change in the locator set significantly increases the length of the code, while the calculations remain in the field $GF(2^m)$.

Description of the Method (Theory in Depth)

A special representation of the parity check matrix H, the generator matrix G of the code, a special selection of the error vector, and/or a special selection of the codeword presentation by the additional field(s) inclusion are utilized. A parity check matrix H is generated for an n, k, d binary generalized (L, G) code wherein n, k, and d, are positive integers, n is a code length, k is a number of information symbols and d is a minimal distance $n \leq \sum_{i=1}^{r} I_{2^m}(i), k \geq n - tm$. Where $I_{2^m}(i)$ is a number of irreducible polynomials of degree i with coefficients from $GF(2^m)$. It is also possible to present transformation $A \times H \times P = H^*$ (or $S \times G \times P = G^*$) as a special permutation of the support set *L*. Therefore, matrix G^* or H^* can be obtained directly, without matrix S or A, and P, from L^* and G(x), where L^* is a special secret permutation of support set L. In such case we can interpret L^* as a second part of a secret key. This can be applied to make changes in the main components of, including, but not limited to, the encryption and signature schemes.

By using the L^* support set directly instead of L with matrix S and P, we can obtain the following variant of McEliece scheme:

Private key: (Decoding algorithm, L^*, G(x))
Public key: G^*
Encryption: Let m be a k-bit message, and let e be a random n-bit vector with Hamming weight $W_H(e) \leq t$. Then $c = m \times G^* \oplus e$ is a ciphertext.

Decryption: Obtaining m by using decoding algorithm (error correcting) with knowledge L^* and G.

In the Niederreiter scheme, by using the two matrices and parity check matrix H, obtained from L and G(x), a public key matrix $H^* = A \times H \times P$ is calculated. As with the McEliece scheme, by using the L^* support set directly instead of L with matrix A and P, we can obtain the following variant of Niederreiter scheme:

Private key: (Decoding algorithm, L^*, G(x))
Public key: H^*
Encryption: Let m be a message, with Hamming weight $W_H(e) \leq t$. Then $c = m \times H^{*T}$ is a ciphertext.
Decryption: Obtaining m by using decoding algorithm (error correcting) with knowledge L^* and G(x).

This implementation allows for: 1) the expansion of the selection of a support set, thereby expanding the available private keys; 2) use of rational functions of degree greater than one to keep the calculation in a finite field with a comparable code length. For example, for rational functions of degree 2 with coefficients from the field GF(2^m), the code length is n = $2^{2m-1} + 2^{m-1}$. The practical benefits of using rational functions with different degree are: 1) reducing the amount of CPU cycles needed in the encryption, decryption, and key generation processes; and 2) increasing the security for codes with the same parameters (n, k, d), as in classical Goppa codes.

The generalized (L, G) code in our algorithm is characterized by a set L where the proper rational functions of $F_{2^m}[x]$ are chosen whose denominators are various irreducible polynomials from $F_{2^m}[x]$ with degree less than or equal r (r > 1), and whose numerators are formal derivatives of the denominators.

In our algorithm, a special support set L is used as a second part of the private secret key in the McEliece and Niederreiter method. In this embodiment, we have the following additional definitions:

Definition #5: Support set L is defined as follows:

$$L = \left\{\frac{f_1'(x)}{f_1(x)}, \frac{f_2'(x)}{f_2(x)}, \ldots, \frac{f_n'(x)}{f_n(x)}\right\},$$

where $f'_i(x)$ is a formal derivative of $f_i(x)$ in GF(2^m) and $f_i(x) = x^{l_i} + c_{l_i-1,i}x^{l_i-1} + \cdots\ c_{1,i}x + c_{0,i}$, $c_{j,i} \in$ GF(2^m), gcd($f_i(x)$, $f_j(x)$)=1, gcd($f_i(x)$, G(x))=1, ∀i, j, i≠ j, deg G(x) = t .

Definition #6: Binary vector a = (a_1, a_2, ..., a_n) is a codeword of generalized (L, G) code if and only if the following equality is satisfied: $\sum_{i=1}^{n} a_i \frac{f_i'(x)}{f_i(x)} = 0\ mod\ G(x)$

For such codes the design bound for the minimum distance: $d_G \geq \frac{2t+1}{l}$, l = max l_i and the decoding algorithm corresponding to it is determined. To construct a parity check matrix for such generalized (*L*, *G*) code the following presentation for rational functions $\frac{f_i'(x)}{f_i(x)}$ by modulo *G(x)* is used:

$$\frac{f'_i(x)}{f_i(x)} = s_i(x) = b_{i,t-1}x^{t-1} + \ b_{i,t-2}x^{t-2} + \cdots . + \ b_{i,1}x^1 + b_{i,0}\ mod\ G(x), b_{i,j}\ \in GF(2^m)$$

The equation for the generalized Goppa code can then be rewritten as:

$$\begin{aligned}\sum_{i=1}^{n} a_i \frac{f'_i(x)}{f_i(x)} &= \sum_{i=1}^{n} a_i\, s_i(x) \\ &= \sum_{i=1}^{n} a_i\, b_{i,t-1}x^{t-1} + \sum_{i=1}^{n} a_i\, b_{i,t-2}x^{t-2} + \cdots . + \sum_{i=1}^{n} a_i\, b_{i,1}x^1 + \sum_{i=1}^{n} a_i\, b_{i,0} \\ &= 0\ mod\ G(x),\end{aligned}$$

From this equation a parity check matrix *H* is obtained:

$$H=\begin{bmatrix} b_{1,t-1} & b_{2,t-1} & \cdots & b_{n,t-1} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1,0} & b_{2,0} & \cdots & b_{n,0} \end{bmatrix}$$

From this parity check matrix we can obtain a generator matrix *G* for the generalized (*L*, *G*) code and by using matrix *S* and *P* to calculate the public key matrix G^{*} =S×G×P.

In another embodiment we can also use the fractions $\frac{f'_i(x)}{f_i(x)}$ with different degrees of $f_i(x)$ for support set *L*. By using irreducible polynomials *f(x)* with degree not greater than r for support set we can obtain a generalized Goppa code with codeword length n ≤ $\sum_{i=1}^{r} I_{2^m}(i)$, where $I_{2^m}(i)$ is a number of irreducible polynomials of degree i with coefficients from GF(2^m).

The following two examples are provided for illustration purposes:

Example 1: In this example m = 6 and r = 2. Since $I_{2^6}(1) = 64, I_{2^6}(2) = \frac{2^{12}-2^6}{2}$ we obtain n = 2048 + 32 = 2080. Let d = 61, t = 30 then we have k ≥ 2080 - 60·6 = 1720.

Example 2: For *l*=2 and $f_i(x) = (x-\beta_i)\left(x-{\beta_i}^{2^m}\right)$, $\beta_i \in GF(2^{2m})\backslash GF(2^m)$, G(x) which is an irreducible polynomial from the polynomial ring $F_{2^m}[x]$. The parity check matrix for this example is:

$$\frac{1}{x-\beta} = \frac{\mathrm{G(x)}-\mathrm{G(\beta)}}{x-\beta} G(\beta)^{-1} \; mod \; G(x)$$

And

$$\frac{\beta+\beta^{2^m}}{(x-\beta)(x-\beta^{2^m})} = \frac{\mathrm{G(x)}-\mathrm{G(\beta)}}{x-\beta} G(\beta)^{-1} + \frac{\mathrm{G(x)}-\mathrm{G}\left(\beta^{2^m}\right)}{x-\beta^{2^m}} G\left(\beta^{2^m}\right)^{-1} \; mod \; G(x)$$

$$\frac{\mathrm{G(x)}-\mathrm{G(\beta)}}{x-\beta} = g_t(x^{t-1}+x^{t-2}\beta+\cdots+\beta^{t-1}) + g_{t-1}(x^{t-2}+x^{t-3}\beta+\cdots+\beta^{t-2}) + \cdots g_2(x+\beta) + g_1 ,$$

where $G(x) = \sum_{i=0}^{t} g_i x^i, g_i \in GF(2^m), g_t \neq 0, g_0 \neq 0$

and

$$\frac{\mathrm{G(x)}-\mathrm{G}\left(\beta^{2^m}\right)}{x-\beta^{2^m}} = g_t\left(x^{t-1}+x^{t-2}\beta^{2^m}+\cdots+\beta^{2^m(t-1)}\right) + g_{t-1}\left(x^{t-2}+x^{t-3}\beta^{2^m}+\cdots+\beta^{2^m(t-2)}\right) + \cdots g_2\left(x+\beta^{2^m}\right) + g_1 ,$$

The coefficients at $x^{t-1}, x^{t-2}, \ldots, x, 1$ in the sum

$$\frac{\mathrm{G(x)}-\mathrm{G(\beta)}}{x-\beta} G(\beta)^{-1} + \frac{\mathrm{G(x)}-\mathrm{G}\left(\beta^{2^m}\right)}{x-\beta^{2^m}} G\left(\beta^{2^m}\right)^{-1}$$

$$x^{t-1}: (\mathrm{G}(\beta)^{-1} + \mathrm{G}\left(\beta^{2^m}\right)^{-1}) g_t ,$$

$$x^{t-2}: (\mathrm{G}(\beta)^{-1} + \mathrm{G}\left(\beta^{2^m}\right)^{-1}) g_{t-1} + (\beta \mathrm{G}(\beta)^{-1} + \beta^{2^m} \mathrm{G}\left(\beta^{2^m}\right)^{-1}) g_t .$$

x^{t-3}: $(\mathrm{G}(\beta)^{-1} + \mathrm{G}(\beta^{2^m})^{-1})g_{t-2} + (\beta\mathrm{G}(\beta)^{-1} + \beta^{2^m}\mathrm{G}(\beta^{2^m})^{-1})g_{t-1} + (\beta^2\mathrm{G}(\beta)^{-1} + \beta^{2\cdot 2^m}\mathrm{G}(\beta^{2^m})^{-1})g_t$,
x^0: $(\mathrm{G}(\beta)^{-1} + \mathrm{G}(\beta^{2^m})^{-1})g_1 + (\beta\mathrm{G}(\beta)^{-1} + \beta^{2^m}\mathrm{G}(\beta^{2^m})^{-1})g_2 + \cdots + (\beta^{t-1}\mathrm{G}(\beta)^{-1} + \beta^{(t-1)\cdot 2^m}\mathrm{G}(\beta^{2^m})^{-1})g_t$.

A parity check matrix H is defined by:
H=

$$\begin{bmatrix} \mathrm{G}(\beta_1)^{-1} + \mathrm{G}(\beta_1{}^{2^m})^{-1} & \mathrm{G}(\beta_2)^{-1} + \mathrm{G}(\beta_2{}^{2^m})^{-1} & \cdots & \mathrm{G}(\beta_n)^{-1} + \mathrm{G}(\beta_n{}^{2^m})^{-1} \\ \beta_1\mathrm{G}(\beta_1)^{-1} + \beta_1{}^{2^m}\mathrm{G}(\beta_1{}^{2^m})^{-1} & \beta_2\mathrm{G}(\beta_2)^{-1} + \beta_2{}^{2^m}\mathrm{G}(\beta_2{}^{2^m})^{-1} & \cdots & \beta_n\mathrm{G}(\beta_n)^{-1} + \beta_n{}^{2^m}\mathrm{G}(\beta_n{}^{2^m})^{-1} \\ & \vdots & & \\ \beta_1{}^{t-1}\mathrm{G}(\beta_1)^{-1} + \beta_1{}^{(t-1)2^m}\mathrm{G}(\beta_1{}^{2^m})^{-1} & \beta_2{}^{t-1}\mathrm{G}(\beta_2)^{-1} + \beta_2{}^{(t-1)2^m}\mathrm{G}(\beta_2{}^{2^m})^{-1} & \cdots & \beta_n{}^{t-1}\mathrm{G}(\beta_n)^{-1} + \beta_n{}^{(t-1)2^m}\mathrm{G}(\beta_n{}^{2^m})^{-1} \end{bmatrix}$$

By way of the foregoing, a special generalization of Goppa codes is constructed with a support set L as a set of rational functions $\frac{f'_i(x)}{f_i(x)}$. The special generalization of Goppa codes is neither a Reed Solomon (RS) code nor an alternant code.

For decoding these generalized Goppa codes, the Goppa polynomial G(x) and support set L must be known. A classical decoding algorithm (Euclidean, Berlekamp-Massey, Patterson, etc.) can then be used.

Using a set of position numerators of degree greater than 1, the degree of Galois field extension *m* for obtaining a support set L is reduced, thereby reducing the complexity of the calculations in the decoding process. The degree *m* of the field extension is reduced by ***r*** times, where ***r*** is the degree of the position numerators.

By way of example, a scheme (2060, 1720, t=30) can be constructed close in parameters to the classical McEliece and Niederreiter (2048, 1718, t=30) scheme by using elements from the Galois field GF(2^6) instead of the field GF(2^{11}) used in the original scheme. Therefore in the scheme of this example, all calculations in the decoding procedure can be done in the Galois field GF(2^6) with only 2^6 elements instead of the Galois field GF(2^{11}) with 2^{11} elements required.

Areas of Application

In the embodiment depicted in Fig. 1, after the cryptographic operations are performed, the result 120 of this computation is returned to the input/output device 106 and output from the cryptographic device 100. This scheme provides a public key cryptographic system and method that can be used to build a highly secure system for data storage, access, encryption, decryption, digital signing, digital signature verification, etc.

Fig. 2 depicts an alternative embodiment of a system in which the foregoing encryption method can be employed. In the system of Fig. 2, the cryptographic engine 116 is implemented on a Chip 130, such as field programmable gate array (FPGA), operating as an independent processing module 132 such as, but not limited to, a TPM-Trusted Platform Module (TPM) or a Universal Serial Bus (USB) Module, rather than being stored in system memory. An advantage of implementing the cryptographic engine 116 on an independent processing module 132 is that the private key 124 is stored on the chip 130 and therefore

separated from the operating system contained in the System Memory 114. This provides an added layer of security as the Private Key 124 is not directly exposed to the file system of the operating system which can be compromised as can the system memory 114.

In the alternative embodiment of Fig. 3, the public key cryptographic device 100 generates a private key 124 and its corresponding public key 126 using an input list of parameters 122. As previously described, the public key 126 (Fig. 3) can be used in encryption and decryption operations. In one embodiment, the public key cryptographic device 100 of Fig. 3 can be implemented as described in connection with Fig. 1 except for the instructions being implemented by the cryptographic engine 116.

An application of the foregoing systems is depicted in Fig. 4, in which the cryptographic engine can be applied to create a Post-Quantum Blockchain (PQBC) 140. In the exemplary embodiment of Fig. 4, the last data block 152 in the PQBC 140 is created by participant node 146 preceded by the data block 150 created by participant node 144 and further preceded by the data block 148 created by participant node 142. All participant nodes use a public key cryptographic device 100 for cryptographic functions. For example, when node 146 creates the last data block 152, node 146 digitally signs the block and records the hash value of the previous data block 150 in the PQBC using a digital signature function in the public key cryptographic device 100. Similarly, the other participant nodes 142 and 144 perform the same steps when creating a new data block. Transaction data inside each of the data blocks can optionally be encrypted using the encryption function in the public key cryptographic device 100. End point security can be further enhanced by employing the system of Fig. 3 in which the private key 124 is maintained separate from the operating system.

Algorithms

Key Generation

Alternative instructions that can be implemented by the device of Fig. 3 are depicted in connection with Fig. 5 in which a method of generating a private and public key pair is shown. The private key 124 and its corresponding public key 126 are used for different functions in the public key cryptographic device 100 such as, but not limited to, encryption, decryption, digital signing, and digital signature verification. The code parameter selection engine 132 chooses m, r, t, n with the property that code length $n \leq \sum_{i=1}^{r} I_{2^m}(i)$, t>r and wherein: m is the degree of the field expansion GF(2^m) in which the operations will be performed during decoding and signature calculations, and which will thereby determine the complexity of the circuit calculations; r is the maximum degree of the denominator of a rational function over $F_{2^m}[x]$ in the set of L; and t is the number of errors in the weighted Hamming metric that can be corrected by the code. There is no limit on how m, r, and t are selected.

For illustration purposes, m determines the Galois field GF(2^m) used in the calculations while r and m determine the size of support set L. Since code length n, r, and t determine a minimal distance of the code, therefore these parameters also determine the number of errors that could be corrected by such error correcting code. The private support set L generator 160 chooses or generates n elements (rational functions $\frac{f'_i(x)}{f_i(x)}$) to support set L. For the sake of clarity, $f_i(x)$ should be an irreducible polynomial of degree r. There are well-known methods

to generate such polynomial, which are outside the scope of this invention. The Private Goppa Polynomial G(x) processor 162 chooses and/or generates primitive polynomial degree t from $F_{2^m}[x]$. For the sake of clarity, G(x) is an irreducible (separable) polynomial of degree t with coefficients from GF(2^m). There are well-known methods to generate such polynomial, any of which can be used. The two elements G and L of Goppa code are defined unequally. Therefore, by using the $(f_i(x))^{-1}$ mod G(x) generator 164 and obtaining $r_i(x)$ using the i-th column $r_i(x)$ of parity check matrix H generator 166, it is now possible to use the i-th binary column of parity check matrix H generator 168 to obtain a number of binary elements equal to the multiplication of t and m (tm), and collect all necessary n columns and tm binary rows $r_{i1}, r_{i2}, \ldots, r_{imt}$ of parity check matrix H from the i-th column $r_i(x) \in GF(2^m)$, deg $r_i(x)$=t-1 during n cycles of the process of steps 160, 164, and 166. For the sake of clarity, $r_i(x) = f'_i(x)\,(f_i(x))^{-1}$ mod G(x), where $f'_i(x)$ is formal derivative of $fi(x)$. The binary parity check matrix H 170 is represented as a tm × n binary matrix. Together with a randomly selected mt x mt non-singular matrix S 172 and another randomly selected n × n permutation matrix P 174 the public key 126 can be obtained by performing a matrix multiplication of $H^* = S \times H \times P$. On the other hand, the private key 124 can be obtained as K = {S, P, L, G(x)}.

Encryption

A method of encrypting a message is depicted in Fig 6 in which the Message 176 is presented as a binary vector e of length n and Hamming weight no more than t/r. The Encrypted Message 178 is obtained as an encrypted vector w of length mt by using matrix multiplication of the public key 126 and the message 176 whereby $w = e \times (H^*)^T$.

Decryption

A method of decrypting an encrypted message is depicted in Fig. 7 in which the encrypted message 180 of length mt is decoded by decoder 182 using a Berlekamp-Massey algorithm or an extended Euclidean algorithm or Patterson algorithm. The private key 124 and the elements of field GF(2^m) 186 are provided to the decoder 182 for the decoding process. For the sake of clarification, in the private key 124 G(x) is an irreducible polynomial of degree t with coefficients from the field GF(2^m) with support set L as a set of rational functions $\frac{f'_i(x)}{f_i(x)}$. The decoded message 184 is an information vector e of the length n and weight in the weighted Hamming metric less than or equal to t.

Signing

A method of obtaining a digital signature for input data, using the cryptographic device 100, is depicted in Fig. 8. This method uses the secret Goppa code elements of a Goppa polynomial G(x) with support set L in a well-known digital signature generation process such as, but not limited to, the Courtois-Finiasz-Sendrier (CFS) signature scheme. Data 190 to be digitally signed is provided to a first hash process 192. The resulting hash value h is used by a second hash process 194 to generate hash value H using h and i (H=hash(h||i) where the length of H is mt bits) where i is a looping incremental value starting at 1. The second hash value H is then used by the decoder 196 in a decoding process based on elements of a Galois field GF(2^m) 198 and a private key 124 using a Berlekamp-Massey or Extended Euclidean algorithm. For the sake of clarification, in the private key 124, G(x) is an irreducible polynomial of degree t with coefficients from the field GF(2^m)) with support set L as a set of rational functions $\frac{f'_i(x)}{f_i(x)}$. The second hash process 194 and the decoder 196 are repeated

with an incrementing i value until a successful decoding is reached. The resulting digital signature 120, represented as {s,i}, consists of two elements: 1) a vector s of the length n and weight in the weighted Hamming metric of less than or equal to t; and 2) a parameter i equal to the number of the successful steps.

Signature Verification

A method of verifying a digital signature for given data, in the cryptographic device 100, is depicted in Fig. 9. In this embodiment, the digital signature 202 to be verified is represented as {s^*,i}. The data 204, which is signed by the digital signature 202, is provided to a hash process 206 as w. Hash process 206 is operated so that the resulting hash value h^*=Hash(h||i), where Hash(w)=h and i is the parameter in the digital signature 202. From the digital signature 202 we can obtain the signature vector s^* of length n and weight less than or equal to t in the weighted Hamming metric. A binary vector h^{**} of the length tm, $h^{**} = s^* \times (H^*)^T$, can be obtained by using matrix multiplication of the public key 126 and the binary vector of the signature s from digital signature 202. The determination between valid signature 208 and an invalid signature 210 can be obtained by comparing the value h^{**} and h^*.

KEM (Key-Encapsulation Mechanism) Compliance

IronCAP supports KEM as specified by NIST and depicted in Fig. 10.

- Communication session is open between the sender and the receiver
- The sender calls IronCAP to obtain the symmetric session key in both plain end encrypted forms
- IronCAP generates a good quality random error vector of n bit length and t/2 weight (e.g. containing t/2 1's), where t is the degree of Goppa polynomial G(x) and 2 - maximal degree of denominators in locator set L (it means that t/2 - the number of errors correctable by code in case when maximal degree of denominators in locator set L is equal 2)
- The random error vector is encrypted by multiplying it with the IronCAP public key matrix obtaining a syndrome vector
- The random error vector is hashed, obtaining a 256-bit KEM session key
- The symmetric session key is equal to KEM session key
- The symmetric session key is returned to the sender for encryption
- IronCAP erases its local copy of the KEM session key
- IronCAP returns the syndrome vector to the sender; this forms a ciphertext, and it is IND-CCA2-compliant
- The ciphertext is sent to the receiver together with the session data encrypted with the symmetric session key
- The symmetric session key is erased by the sender
- The receiver passes the ciphertext to IronCAP for decryption
- IronCAP decrypts the syndrome vector with the IronCAP private key obtaining the random error vector
- The random error vector is hashed and obtaining the 256-bit KEM session key
- The symmetric session key is equal to KEM session key
- IronCAP returns the symmetric session key
- The receiver decrypts the session data using the symmetric session key
- The symmetric session key and KEM session key are erased by the receiver

- The communication session is closed.

In addition, IronCAP provides a "legacy" KEM variant that is best suitable for using with the existing popular cryptographic frameworks.

Since the GnuPG and PKCS#11 frameworks that IronCAP is used with are not fully KEM-complaint in the way how NIST is describing KEM, e.g. they are generating good quality random symmetric (AES, etc.) keys for every message outside of the public key (RSA, ECC, ICC, etc) modules and letting the public key modules encrypt-decrypt them employing padding techniques whenever required, we must adapt the KEM procedure to this practical reality.

Here and further, a "symmetric session key" means a good quality random key generated with a symmetric (AES, etc.) key module for session data encryption/decryption and passed to IronCAP for encryption; a "KEM session key" means a good quality random key generated with IronCAP for encryption/decryption of the symmetric session key.

Our "legacy" KEM variation is implemented as follows, depicted in Fig. 11

- Communication session is open between the sender and the receiver
- A symmetric session key is generated by the sender
- The symmetric session key is passed to IronCAP for encryption
- IronCAP is generating a good quality random error vector of n bit length and t/2 weight (e.g. containing t/2 1's), where t is the degree of Goppa polynomial G(x) and 2 - maximal degree of denominators in locator set L (it means that t/2 - the number of errors correctable by code in case when maximal degree of denominators in locator set L is equal 2)
- The random error vector is encrypted by multiplying it with the IronCAP public key matrix obtaining a syndrome vector
- The random error vector is hashed, obtaining a 256-bit KEM session key
- The symmetric session key is encrypted with the KEM session key by XOR'ing; this is eliminating the known padding issues, as the length of the supported by IronCAP symmetric session keys is always a multiple of 256 bit
- IronCAP erases its local copy of the KEM session key
- IronCAP returns the encrypted symmetric session key and the syndrome vector; these form a ciphertext, and it is IND-CCA2-compliant
- The ciphertext is sent to the receiver together with the session data encrypted with the symmetric session key
- The symmetric session key is erased by the sender
- The receiver passes the ciphertext to IronCAP for decryption
- IronCAP decrypts the syndrome vector with the IronCAP private key obtaining the random error vector
- The random error vector is hashed and obtaining the 256-bit KEM session key
- The symmetric session key is decrypted by XOR'ing with the 256-bit KEM session key
- IronCAP returns the decrypted symmetric session key and erases the KEM session key
- The receiver decrypts the session data using the symmetric session key
- The symmetric session key is erased by the receiver
- Communication session is closed.

Brief Description of Figures

FIG.1 is a block diagram of a system for carrying out a cryptographic method where the system is used for encryption, decryption, generating digital signatures, verifying digital signatures, etc.

FIG.2 is a block diagram of a system for carrying out a cryptographic method where the system provides further security through use of a trusted platform module (TPM) or a universal serial bus (USB) interface.

FIG.3 is a block diagram of a system for carrying out a cryptographic method where the system generates a public key and a private key based on a set of input parameters.

FIG.4 is a block diagram of a system for carrying out a cryptographic method where a Post-Quantum Blockchain (PQBC) is created so that data security of the PQBC is strengthened against cyber attacks from quantum computers and classical computers.

FIG.5 is a flowchart illustrating a process of generating a private key and a corresponding public key in a public key cryptographic device.

FIG.6 is a flowchart illustrating the process of encrypting data using a public key in the public key cryptographic device.

FIG.7 is a flowchart illustrating the process of decrypting an encrypted data using a corresponding private key in a public key cryptographic device.

FIG.8 is a flowchart illustrating the process of digital signing data using a private key in a public key cryptographic device.

FIG.9 is a flowchart illustrating the process of verifying a digital signature using a corresponding public key in a public key cryptographic device.

FIG.10 is a flowchart illustrating the KEM-compliant process of encrypting a communication session between a sender and a receiver.

FIG.11 is a flowchart illustrating the KEM-compliant process of encrypting a communication session between "legacy" senders and receivers.

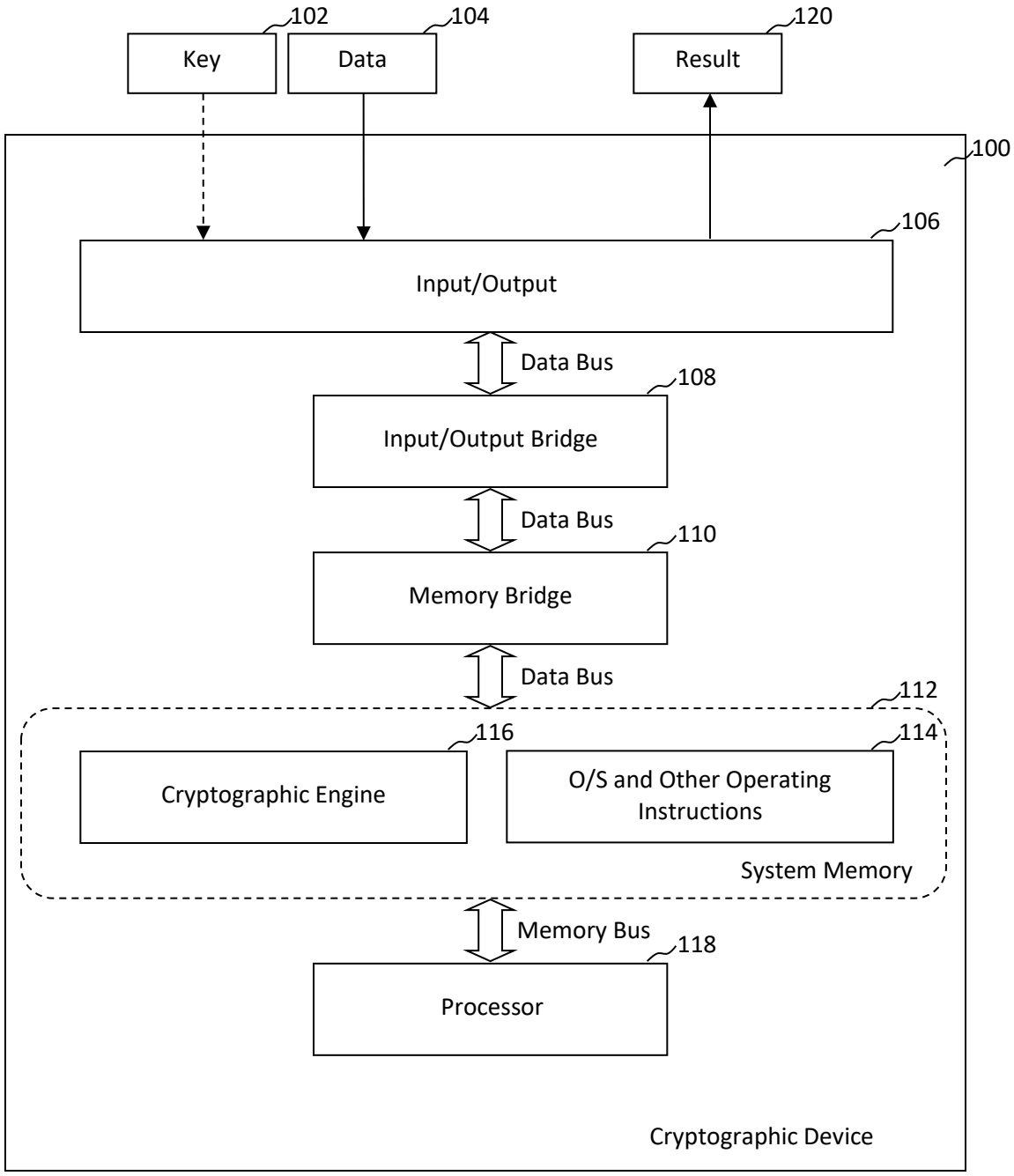
102
104
120
Key
Data
Result
100
106
Input/Output
Data Bus
108
Input/Output Bridge
Data Bus
110
Memory Bridge
Data Bus
112
116
114
Cryptographic Engine
O/S and Other Operating Instructions
System Memory
Memory Bus
118
Processor
Cryptographic Device

Fig. 1

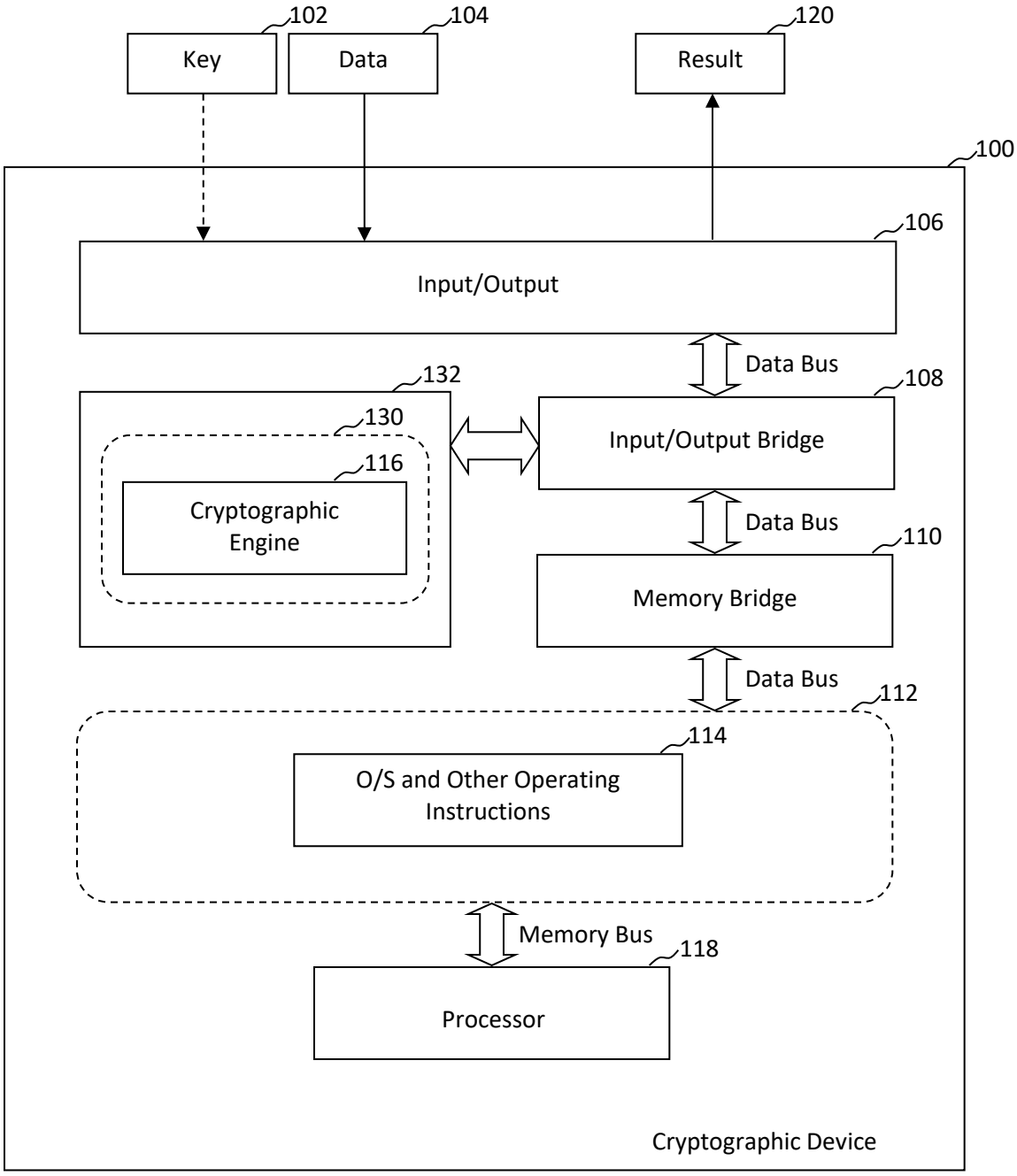
102
104
120
Key
Data
Result
100
106
Input/Output
Data Bus
132
108
130
Input/Output Bridge
116
Cryptographic Engine
Data Bus
110
Memory Bridge
Data Bus
112
114
O/S and Other Operating Instructions
Memory Bus
118
Processor
Cryptographic Device

Fig. 2

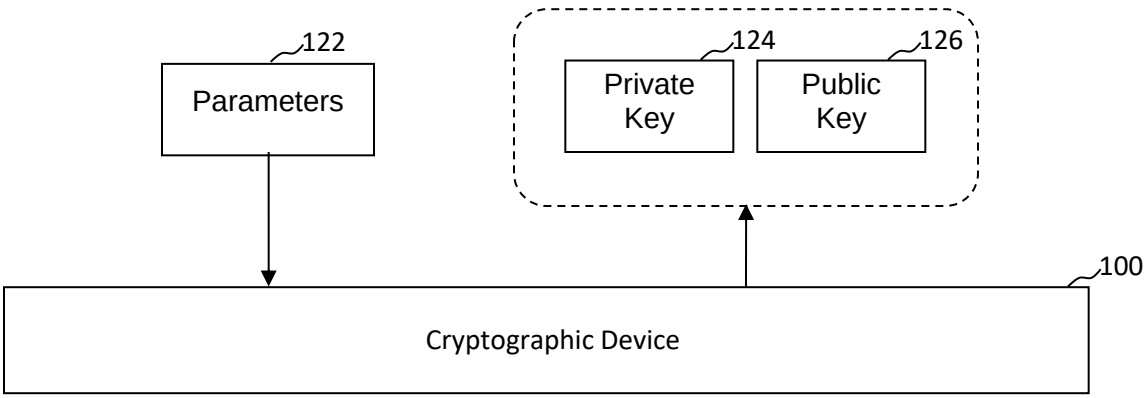
122
Parameters
124
Private Key
126
Public Key
100
Cryptographic Device

Fig. 3

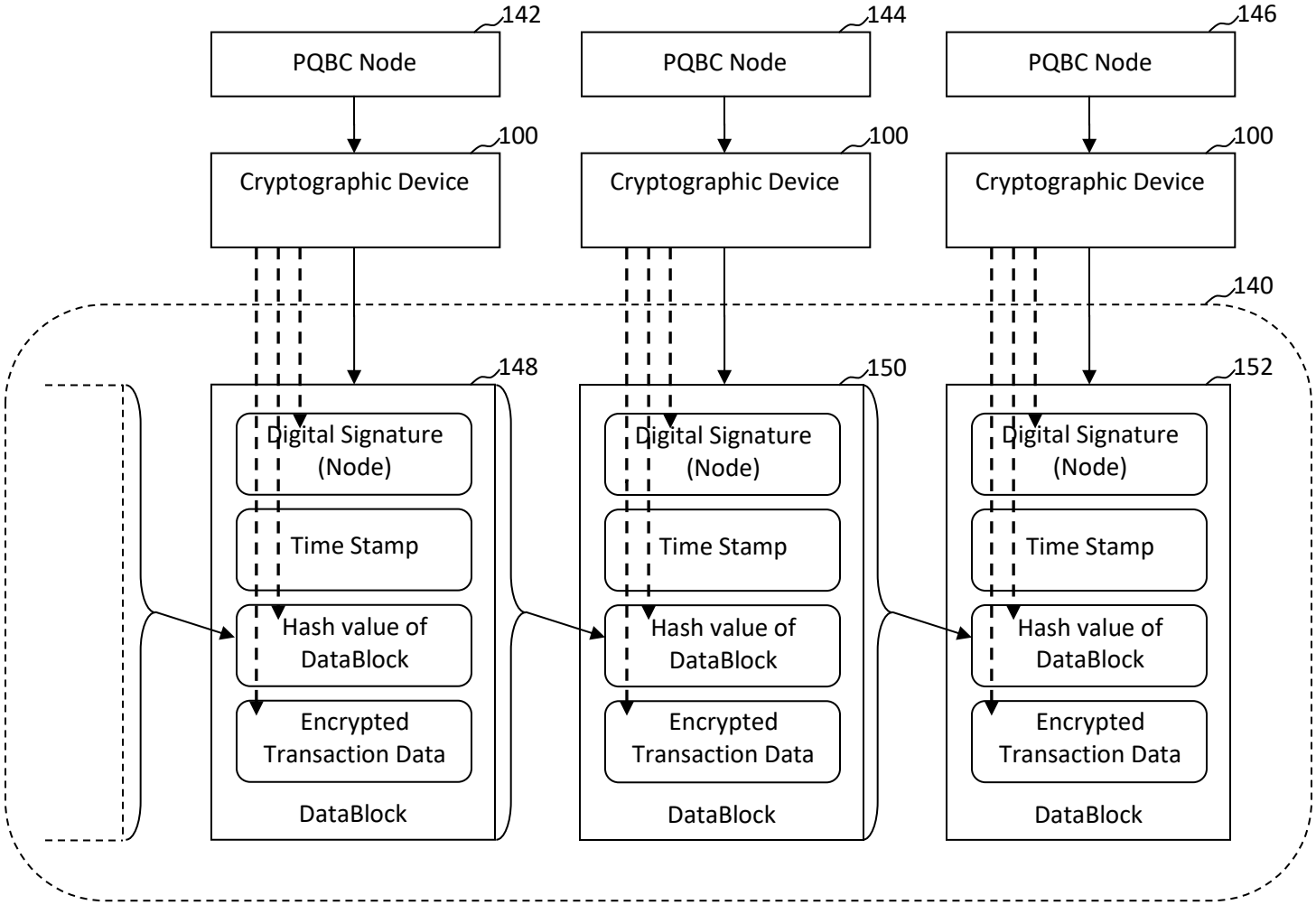
142
PQBC Node
144
PQBC Node
146
PQBC Node
100
Cryptographic Device
100
Cryptographic Device
100
Cryptographic Device
140
148
150
152
Digital Signature (Node)
Time Stamp
Hash value of DataBlock
Encrypted Transaction Data
DataBlock
Digital Signature (Node)
Time Stamp
Hash value of DataBlock
Encrypted Transaction Data
DataBlock
Digital Signature (Node)
Time Stamp
Hash value of DataBlock
Encrypted Transaction Data
DataBlock

Fig. 4

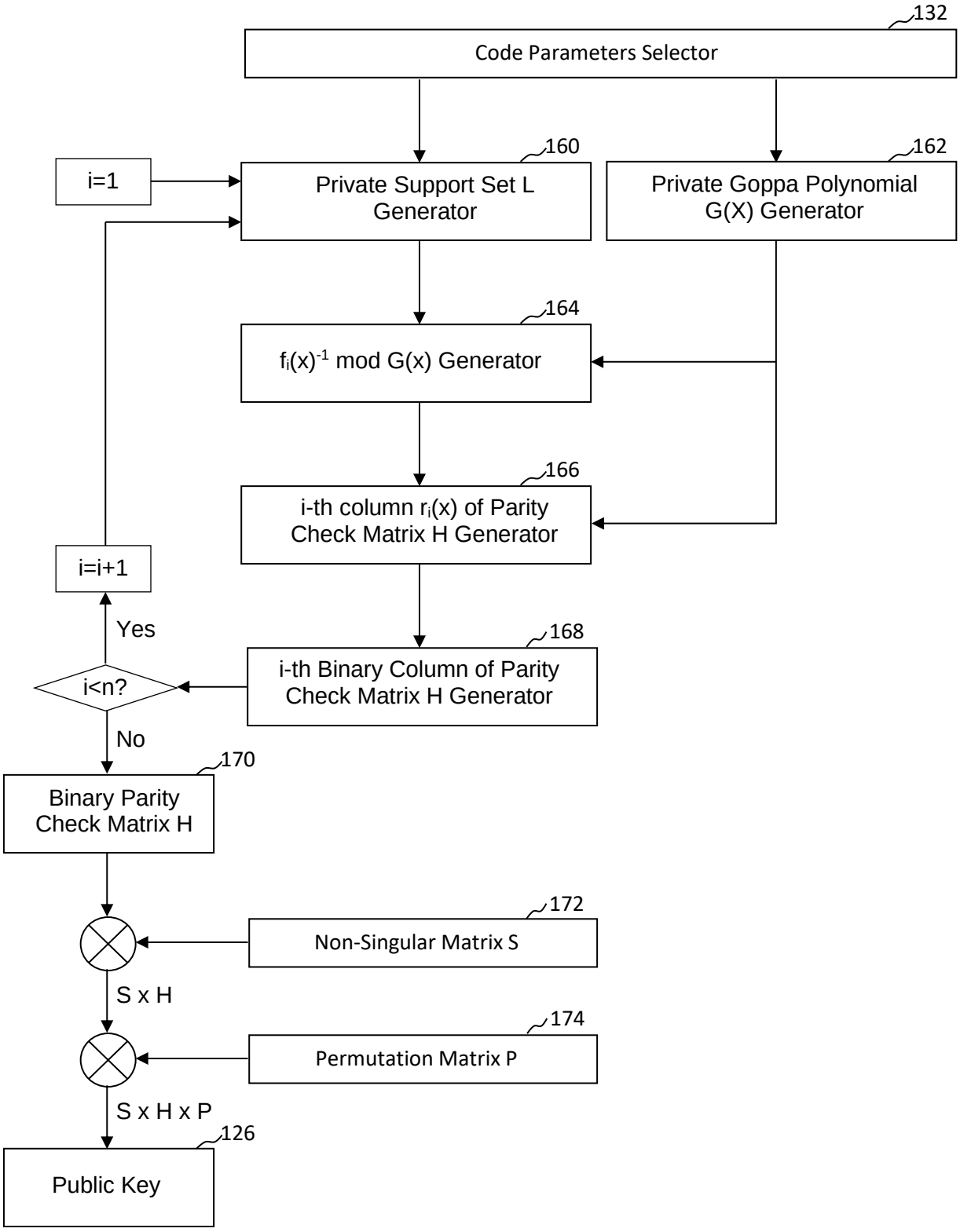
132
Code Parameters Selector
160
162
i=1
Private Support Set L Generator
Private Goppa Polynomial G(X) Generator
164
$f_i(x)^{-1}$ mod G(x) Generator
166
i-th column $r_i(x)$ of Parity Check Matrix H Generator
i=i+1
Yes
168
i-th Binary Column of Parity Check Matrix H Generator
i<n?
No
170
Binary Parity Check Matrix H
172
Non-Singular Matrix S
S x H
174
Permutation Matrix P
S x H x P
126
Public Key

Fig. 5

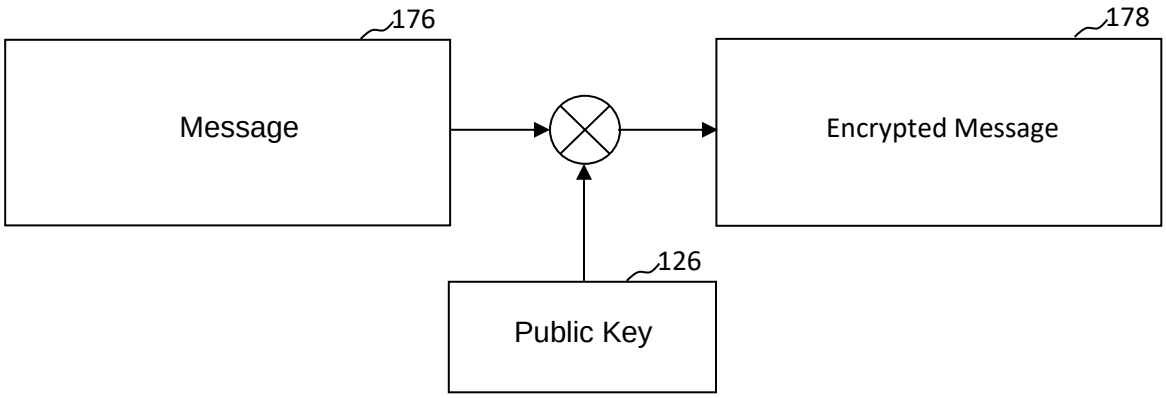
176
Message
178
Encrypted Message
126
Public Key

Fig. 6

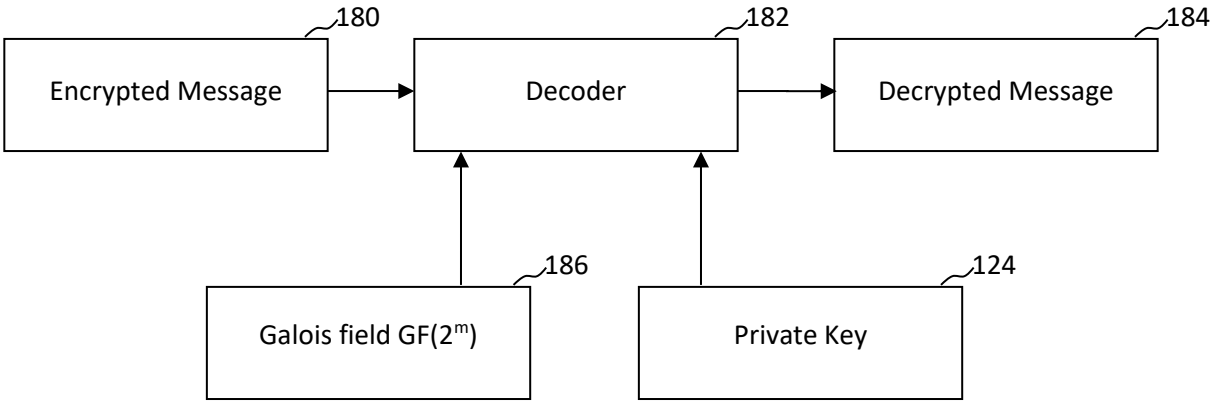
180
Encrypted Message
182
Decoder
184
Decrypted Message
186
Galois field GF(2^m)
124
Private Key

Fig. 7

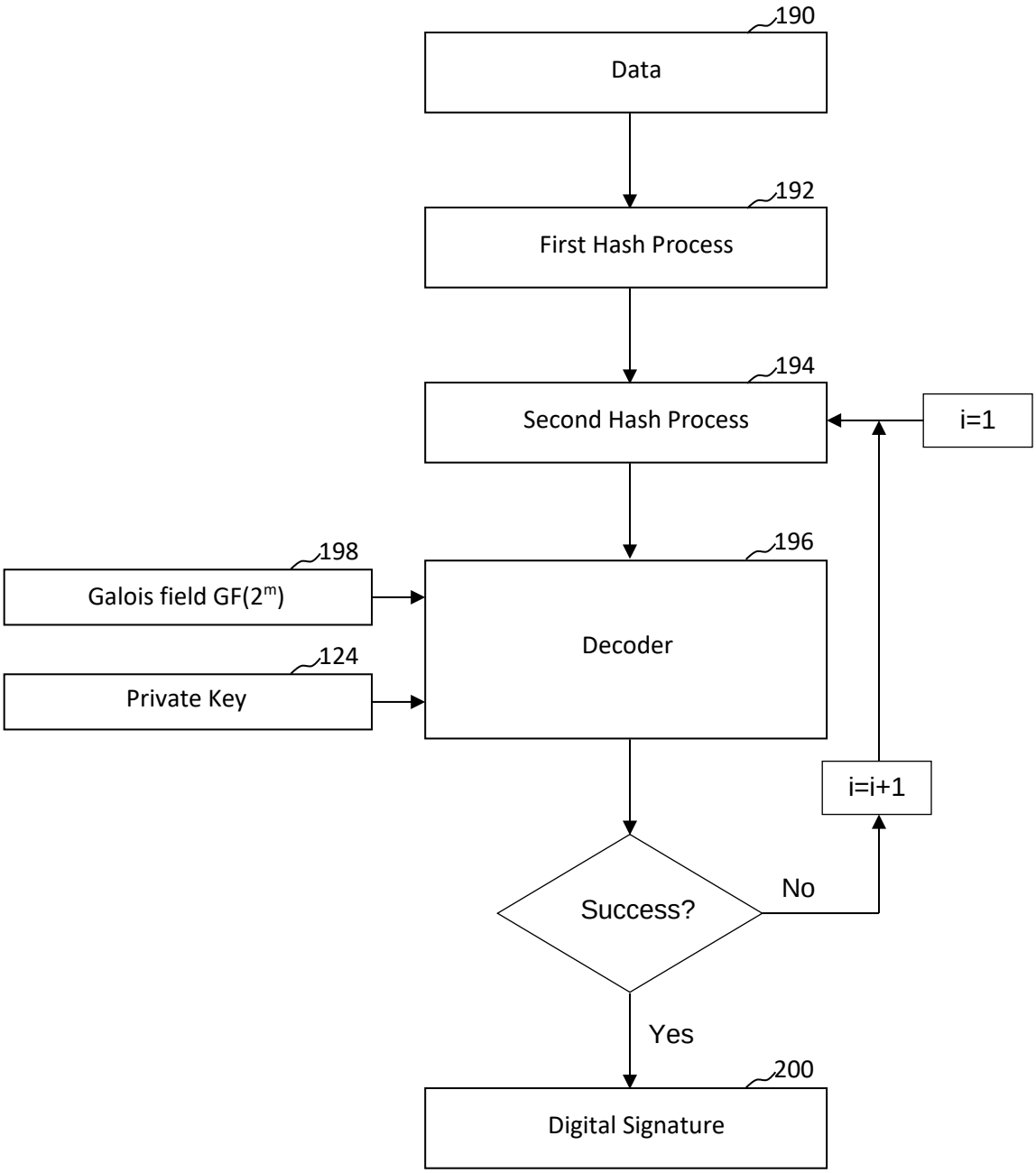
190
Data
192
First Hash Process
194
Second Hash Process
i=1
198
Galois field GF(2^m)
196
Decoder
124
Private Key
i=i+1
Success?
No
Yes
200
Digital Signature

Fig. 8

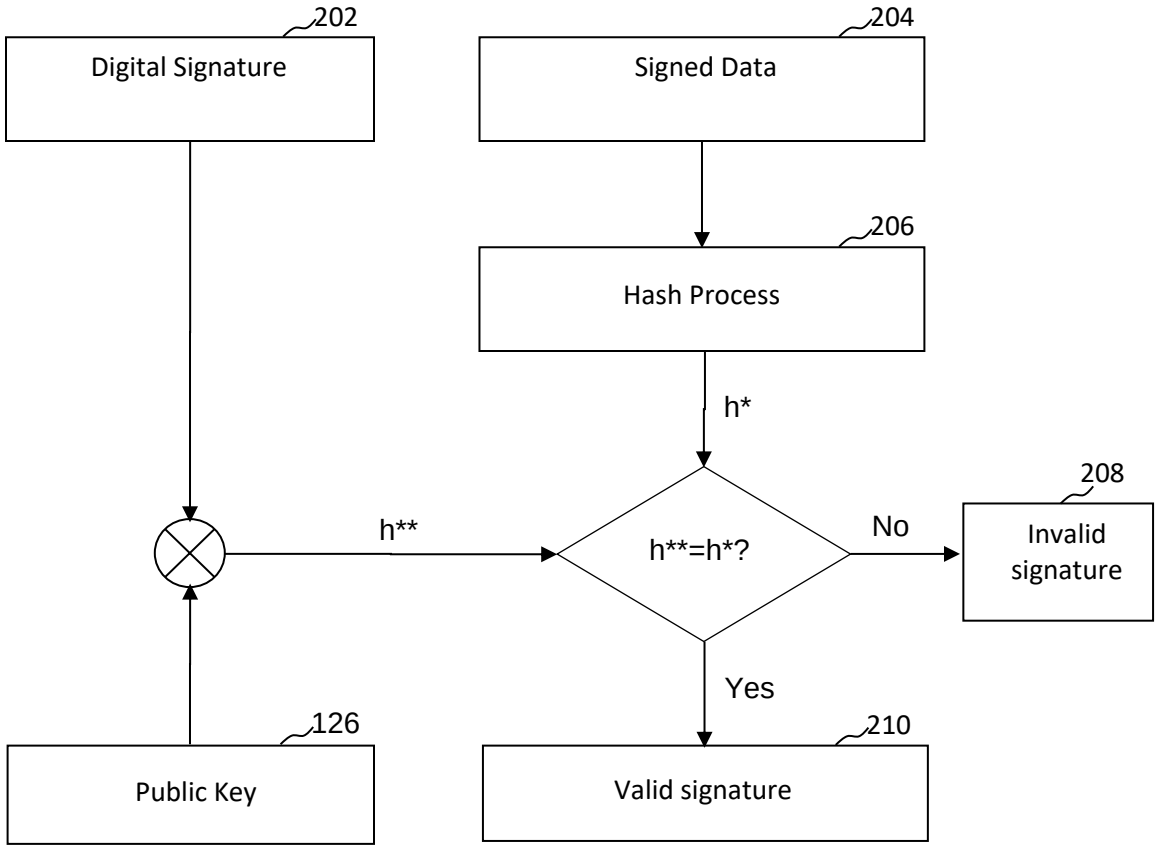
202
Digital Signature
204
Signed Data
206
Hash Process
h*
208
h**
h**=h*?
No
Invalid signature
Yes
126
Public Key
210
Valid signature

Fig. 9

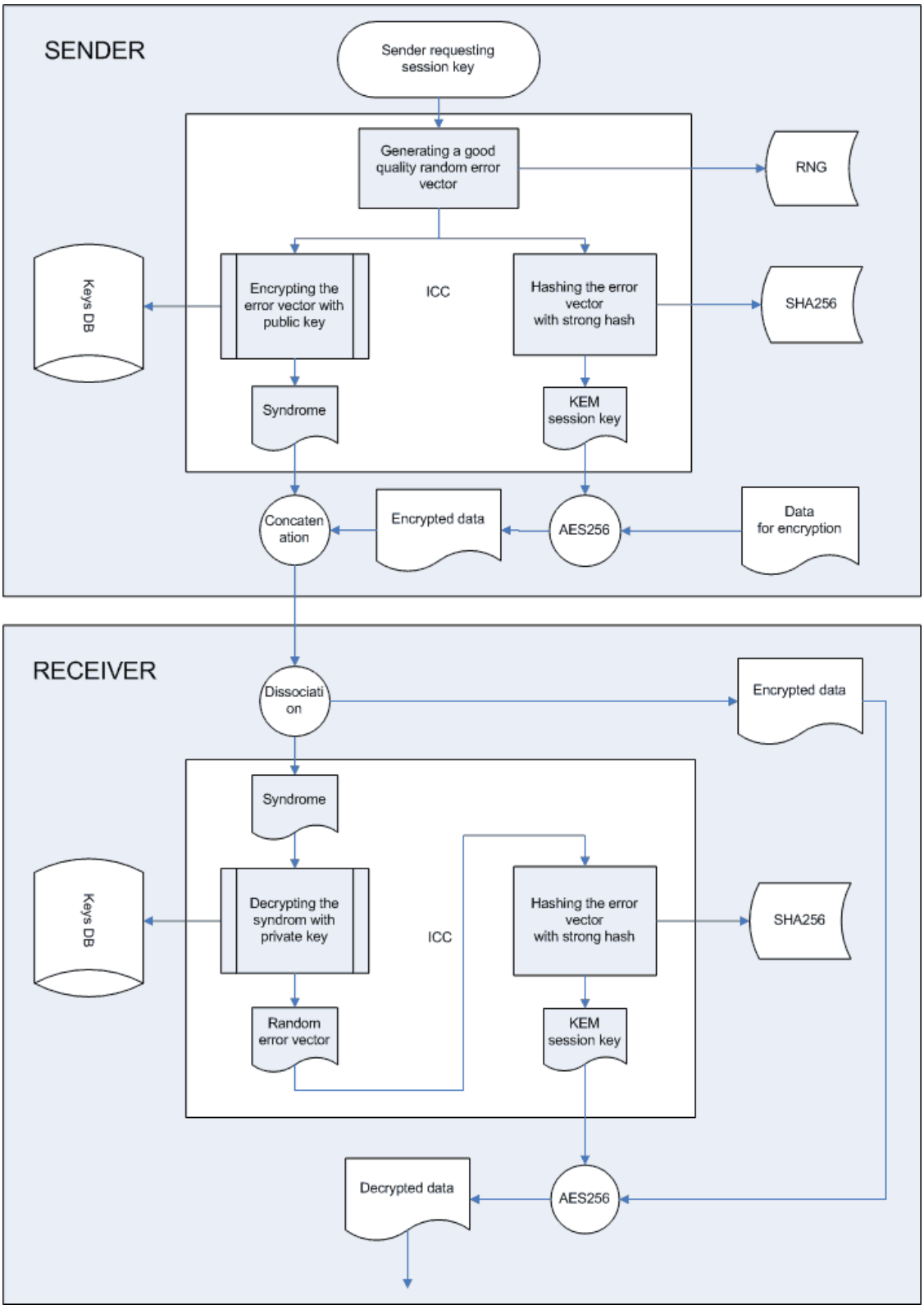
SENDER
Sender requesting session key
Generating a good quality random error vector
RNG
Keys DB
Encrypting the error vector with public key
ICC
Hashing the error vector with strong hash
SHA256
Syndrome
KEM session key
Concaten ation
Encrypted data
AES256
Data for encryption
RECEIVER
Dissociati on
Encrypted data
Syndrome
Keys DB
Decrypting the syndrom with private key
ICC
Hashing the error vector with strong hash
SHA256
Random error vector
KEM session key
Decrypted data
AES256

Fig. 10

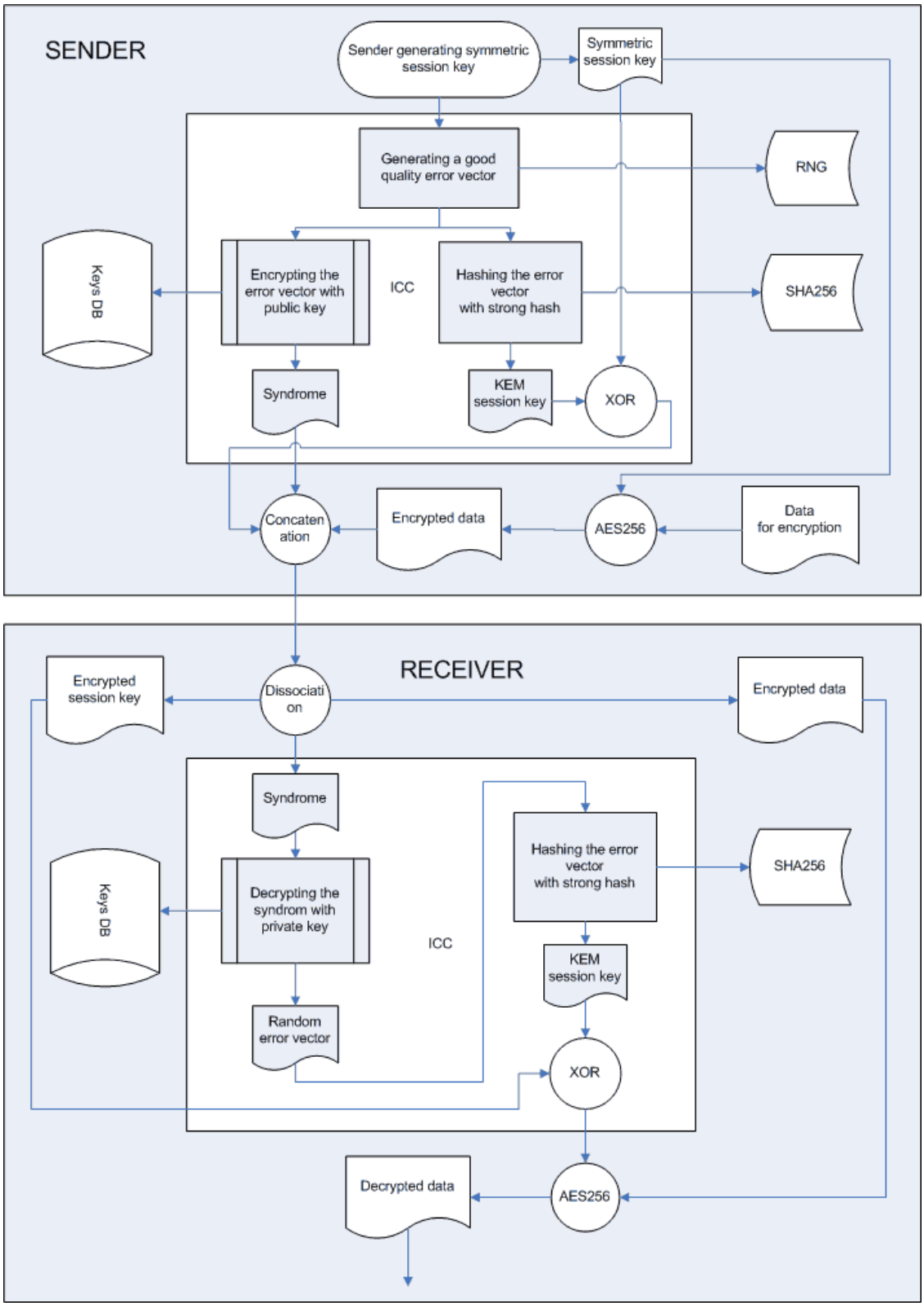
SENDER
Sender generating symmetric session key
Symmetric session key
Generating a good quality error vector
RNG
Keys DB
Encrypting the error vector with public key
ICC
Hashing the error vector with strong hash
SHA256
Syndrome
KEM session key
XOR
Concaten ation
Encrypted data
AES256
Data for encryption
RECEIVER
Encrypted session key
Dissociati on
Encrypted data
Syndrome
Keys DB
Decrypting the syndrom with private key
ICC
Hashing the error vector with strong hash
SHA256
KEM session key
Random error vector
XOR
Decrypted data
AES256

Fig. 11

Tomorrow's Cyber Security, Today
IRONCAP

About 01 Communique

Established in 1992, 01 Communique is always at the forefront of technology. Its latest innovation is on cyber security with its latest development focused on Post-Quantum Cryptography (PQC). Our patented invention PQC, together with PQC selected by NIST, are designed to operate on today's conventional computer systems to safeguard against potential cyber attacks from quantum computers. Our technology has been designed to transform today's cyber security in a way that is safe against future attacks from the world of quantum computers. Examples of vertical applications are emails/files encryption, digital signatures, blockchain security, remote access/VPN, password management, credit card security, cloud storage, artificial intelligence, IoT and web site security.