CHAPTER 14

Lightweight Cryptography and Post-Quantum Cryptography

LEARNING OBJECTIVES

After studying this chapter, you should be able to

- Explain the concept of embedded system.
- Explain the concept of constrained device.
- Give a presentation on the concept of lightweight cryptography and the types of cryptographic algorithms for which lightweight cryptography is of interest.
- Discuss the constraints that affect the design of lightweight cryptographic algorithms.
- Discuss the security requirements for lightweight cryptographic algorithms.
- Present an overview of approaches to lightweight cryptography for authenticated encryption, hash functions, and message authentication codes.
- Explain the need for post-quantum cryptographic algorithms and which types of algorithms are affected.
- Present an overview of mathematical approaches to developing post-quantum cryptographic algorithms.

Two recent areas of strong interest in the field of cryptography are lightweight cryptography and post-quantum cryptography. It is likely in the coming years that a number of new algorithms in both areas will be widely deployed. In essence, lightweight cryptography is focused on developing algorithms that, while secure, minimize execution time, memory usage, and power consumption. Such algorithms are suitable for small embedded systems such as those in wide use in the Internet of Things (IoT). Work on lightweight cryptography is almost exclusively devoted to symmetric (secret key) algorithms and cryptographic hash functions.

Post-quantum cryptography is an area of study that arises from the concern that quantum computers would be able to break currently used asymmetric cryptographic algorithms. Shor's algorithm demonstrated a feasible way to break asymmetric algorithms that rely on either integer factorization or discrete logarithms. Thus, work on post-quantum cryptography is devoted to developing new asymmetric cryptographic algorithms.

14.1 LIGHTWEIGHT CRYPTOGRAPHY CONCEPTS

Lightweight cryptography is a subfield of cryptography concerned with the development of cryptographic algorithms for resource-constrained devices. The term **lightweight** refers to the characteristic that a cryptographic algorithm makes minimal resource demands on the host system. For many existing cryptographic standards, the algorithms incorporate trade-offs between security, performance, and cost requirements that make them unsuitable for implementation in resource-constrained devices. Lightweight cryptography includes attempts to develop efficient implementations of conventional cryptographic algorithms as well as the design of new lightweight algorithms.

Embedded Systems

The term **embedded system** refers to the use of electronics and software within a product that has a specific function or set of functions, as opposed to a general-purpose computer, such as a laptop or desktop system. We can also define an embedded system as any device that includes a computer chip, but that is not a general-purpose workstation, desktop, or laptop computer. Hundreds of millions of computers are sold every year, including laptops, personal computers, workstations, servers, mainframes, and supercomputers. In contrast, tens of billions of microcontrollers are produced each year that are embedded within larger devices. Today, many, perhaps most, devices that use electric power have an embedded computing system. It is likely that in the near future, virtually all such devices will have embedded computing systems.

Types of devices with embedded systems are almost too numerous to list. Examples include cell phones, digital cameras, video cameras, calculators, microwave ovens, home security systems, washing machines, lighting systems, thermostats, printers, various automotive systems (e.g., transmission control, cruise control, fuel injection, anti-lock brakes, and suspension systems), tennis rackets, toothbrushes, and numerous types of sensors and actuators in automated systems.

Microcontrollers A **microcontroller** is a single chip that contains the processor, nonvolatile memory for the program (ROM or flash), volatile memory for input and output (RAM), a clock, and an I/O control unit. It is also called a "computer on a chip." A microcontroller chip makes a substantially different use of the logic space available. The processor portion of the microcontroller has a much lower silicon area than other microprocessors and much higher energy efficiency.

Billions of microcontroller units are embedded each year in myriad products from toys to appliances to automobiles. For example, a single vehicle can use 70 or more microcontrollers. Typically, especially for the smaller, less expensive microcontrollers, they are used as dedicated processors for specific tasks. For example, microcontrollers are heavily utilized in automation processes. By providing simple reactions to input, they can control machinery, turn fans on and off, open and close valves, and so forth. They are integral parts of modern industrial technology and are among the most inexpensive ways to produce machinery that can handle extremely complex functionalities.

Microcontrollers come in a range of physical sizes and processing power. Processors range from 4-bit to 32-bit architectures. Microcontrollers tend to be much slower than microprocessors, typically operating in the MHz range rather than the GHz speeds of microprocessors. Another typical feature of a microcontroller is that it does not provide for human interaction. The microcontroller is programmed for a specific task, embedded in its device, and executes as and when required.

Deeply Embedded Systems A subset of embedded systems, and a quite numerous subset, is referred to as **deeply embedded systems**. In general terms, a deeply embedded system has a processor whose behavior is difficult to observe both by the programmer and the user. A deeply embedded system uses a microcontroller, is not programmable once the program logic for the device has been burned into ROM (read-only memory), and has no interaction with a user.

Deeply embedded systems are dedicated, single-purpose devices that detect something in the environment, perform a basic level of processing, and then do something with the results. Deeply embedded systems often have wireless capability and appear in networked configurations, such as networks of sensors deployed over a large area (e.g., factory, agricultural field). The IoT depends heavily on deeply embedded systems. Typically, deeply embedded systems have extreme resource constraints in terms of memory, processor size, time, and power consumption.

Constrained Devices

A **constrained device** is a device with limited volatile and nonvolatile memory, limited processing power, and a low data rate transceiver. Many devices in the IoT, particularly the smaller, more numerous devices, are resource constrained. As pointed out in [SEGH12], technology improvements following Moore's law continue to make embedded devices cheaper, smaller, and more energy-efficient but not necessarily more powerful. Typical constrained devices are equipped with 8- or 16-bit microcontrollers that possess very little RAM and storage capacities. Resource-constrained devices are often equipped with an IEEE 802.15.4 radio, which enables low-power low-data-rate wireless personal area networks (WPANs) with data rates of 20–250 kbps and frame sizes of up to 127 octets.

RFC 7228 (Terminology for Constrained-Node Networks) defines three classes of constrained devices (Table 14.1):

- **Class 0:** These are very constrained devices, typically sensors, called *motes*, or *smart dust*. Motes can be implanted or scattered over a region to collect data and pass it on from one to another to some central collection point. For example, a farmer, vineyard owner, or ecologist could equip motes with sensors that detect temperature, humidity, etc., making each mote a mini weather station. Scattered throughout a field, vineyard or forest, these motes would allow the tracking of microclimates. Class 0 devices generally cannot be secured or managed comprehensively in the traditional sense. They will most likely be preconfigured (and will be reconfigured rarely, if at all) with a very small data set.
- **Class 1:** These are quite constrained in code space and processing capabilities, such that they cannot easily talk to other Internet nodes employing a full protocol stack. However, they are capable enough to use a protocol stack specifically designed for constrained nodes and participate in meaningful conversations without the help of a gateway node.
- **Class 2:** These are less constrained and fundamentally capable of supporting most of the same protocol stacks as used on notebooks or servers. However, they are still very constrained compared to high-end IoT devices. Thus, they require lightweight and energy-efficient protocols and low transmission traffic.

Table 14.1 Classes of Constrained Devices

Class	Data Size (RAM)	Code Size (flash, ROM)
Class 0	« 10 kB	« 100 kB
Class 1	~ 10 kB	~ 100 kB
Class 2	~ 50 kB	~ 250 kB

Categories of Constraints for Lightweight Cryptography

It is useful to define the specific constraints that relate to the design of **lightweight cryptographic algorithms**. ISO 29192-1 (*Lightweight Cryptography—Part 1: General*, June 2012) lists the following as the key constraints:

- **Chip area:** Chip area is of concern when a cryptographic algorithm is implemented in hardware. Very small devices, such as small sensors, have limited available chip area to provide for security. Typically, chip area is expressed in gate equivalents (GEs). The GE value is derived by dividing the area of the integrated circuit by the area of a two-input NAND gate in the appropriate technology.
- **Energy consumption:** Many constrained devices operate from a very small battery or energy derived from an incoming signal. Accordingly, algorithms may need to be designed to minimize energy consumption. Energy consumption is a function of several factors including the processing time, the chip area (when implemented in hardware), the operating frequency, and the number of bits transmitted between entities (in wireless transmissions in particular).
- **Program code size and RAM size:** Constrained devices typically have very limited space for program code (e.g., in ROM) and RAM needed for execution. Thus, cryptographic algorithms need to be compact in terms of code and make use of minimal RAM during execution.
- **Communications transmission rate:** Very constrained devices, such as sensors and RFID tags, may be capable of very limited data rates. Thus, the amount of security-related data that needs to be transmitted, such as message authentication codes and key exchange material, needs to be extremely small.
- **Execution time:** For some devices, such as contactless cards and RFID tags, execution time is constrained by the amount of time the device is present in the communication zone.

Security Considerations for Various Applications

Security requirements vary for different types of constrained devices. A useful list of application areas is defined by the CRYPTREC[1] Lightweight Cryptography Working Group in [CRYP17]. The following section summarizes key considerations for these devices.

Radio-Frequency Identification (RFID) RFID is a data collection technology that uses electronic tags attached to items to allow the items to be identified and tracked by a remote system. RFID technology is increasingly becoming an enabling technology for IoT. The main elements of an RFID system are tags and readers. RFID tags are small programmable devices, with an attached antenna, used for object, animal, and human tracking. They come in a variety of shapes, sizes, functionalities, and costs. RFID readers acquire and sometimes rewrite information stored on RFID tags that come within operating range (a few inches up to several feet). Readers are usually connected to a computer system that records and formats the acquired information for further uses.

[1]CRYPTREC is the Cryptography Research and Evaluation Committee created by the Japanese Government to evaluate and recommend cryptographic techniques for government and industrial use.

RFID devices require cryptographic algorithms that use a very small amount of logic and memory. Despite this, depending on the use of the RFID tag, a number of security mechanisms may be required. [SAAR12] lists the following as example uses and the corresponding security requirements:

- **Counterfeit goods:** RFID tags can be cloned or modified in order for counterfeit products or parts to pass as genuine. Authentication can counter this threat.
- **Environmental logging:** Tampering with information such as temperature logs can pose a threat to the supply chain management of products such as fresh goods and medical supplies. Data and device authentication can counter this threat.
- **Privacy of Electronic Product Code (EPC):** The EPC is designed to be stored on an RFID tag and it provides a universal identifier for every physical object anywhere in the world. This raises serious privacy issues if such tags are attached to personal items. Therefore, the tag must also identify the reader as trusted before divulging traceable information.
- **Antitheft:** Data may be written to the tag to indicate to an exit portal whether or not that item has been sold. Persistent memory write and lock operations must be protected to prevent theft.
- **Returns:** When a tag is returned to a store or manufacturer, an authenticated reset/write mechanism allows it to be reused. The tags maintain some amount of persistent memory; read, write, and lock operations to this memory must be authenticated to prevent tamper and unauthorized modification. Authenticated reads allow data to be visible only for the tag's owner.

Electronic Home Appliances and Smart TV A number of home appliances, including air conditioners, ovens, and televisions, are now equipped with embedded processors that provide a range of services and may be connected to the Internet. To lower cost, these embedded systems are generally very constrained and are almost constantly under full load, leaving limited resources for security features. These devices are vulnerable to unauthorized access that may tamper with the control signals or issue illegal commands that would lead to abnormal operations. These devices will also usually have updateable software. Thus, authentication methods are important.

Smart Agricultural Sensors Environmental sensors in agricultural settings can improve productivity and yield. For example, the sensors can operate with actuators to control the timing and amount of watering and to automatically open and close greenhouse windows and to schedule pest control. Requirements for sensor networks include autonomously driven, small size, low power consumption, and low cost so that large numbers of sensors can be employed. These devices need to be tamper resistant to prevent sabotage.

Medical Sensors Wireless medical sensors permit health monitoring of patients outside of a hospital setting, capturing and transmitting a number of medical and health-related measures. These devices, particularly if that are implanted, are generally extremely small and use very little power.

Industrial Systems In factories, the transportation, processing, and assembly operations have been automated to improve operational efficiency. Several machine

tools and robots can be connected by a network to share manufacturing information and to manage the processes based on the data collected by sensors. Through a network, it is also possible to store information at a single place and to manage the equipment from a central location.

When connected to the Internet, these systems can be vulnerable both to the exposure of data and to sabotage. The risk is especially high in the case of critical public infrastructure, such as power distribution systems, nuclear power plants, water treatment, and air traffic control. The execution of unauthorized commands or the failure to execute authorized commands can lead to significant and even catastrophic damage. Thus, authentication, authorization, and availability mechanism are essential.

Automobiles Modern automobiles provide both in-vehicle communication as well as wireless communication with external entities via small embedded systems. These onboard embedded devices are part of what are termed vehicle communications systems, which are networks in which vehicles and roadside units are the communicating nodes, providing each other with information, such as safety warnings and traffic information. They can be effective in avoiding accidents and traffic congestion.

Among security concerns are authentication to ensure that all the communications are accurate and can't be spoofed, and privacy to ensure that the communications can't be used to track cars [NHTS14].

RFC 7744 provides additional examples of uses of constrained devices and their security requirements.

Design Trade-Offs

Figure 14.1 illustrates the trade-offs between security, cost, and performance in designing lightweight cryptographic algorithms. In general terms, for any given algorithm, the longer the key and the more rounds, the greater the security.

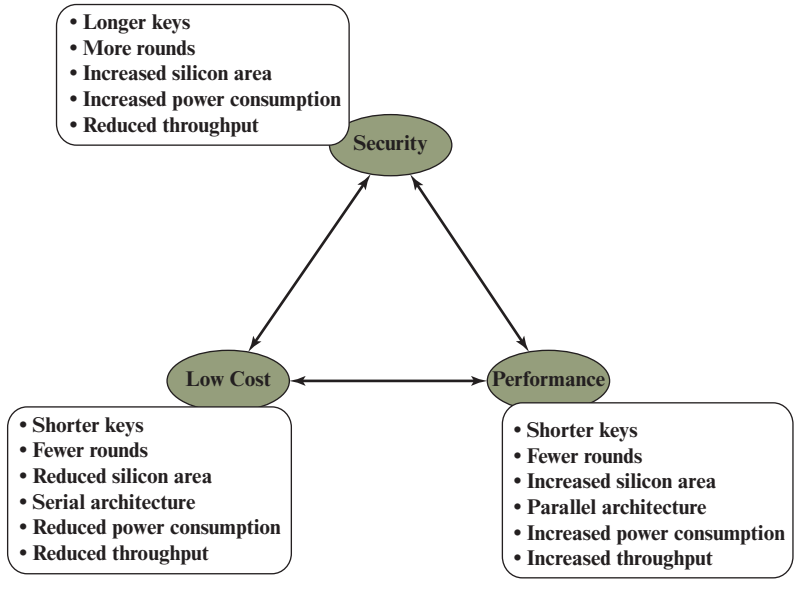


Figure 14.1 Lightweight Cryptography Trade-Offs

This implies a reduced throughput, in terms of the amount of plaintext processed per time unit, as well as increased power consumption. Similarly, the more complex an algorithm or its implementation, the more security it can provide, but this generally requires increased silicon area, either for hardware implementation or software implementation.

Thus, achieving greater security can degrade either cost or performance objectives, or both. As between performance and cost, there is also a trade-off in terms of the architecture, with a serial architecture generally providing lower cost, but a parallel architecture providing greater performance.

Security Requirements

ISO 29192 defines a minimum security strength for lightweight cryptography of 80 bits. The standard defines the security strength to be the number associated with the amount of work (i.e., the number of operations) that is required to break a cryptographic algorithm or system. A security strength of n implies that the required workload of breaking the cryptosystem is equivalent to $2n$ executions of the cryptosystem. Most standards documents recommend a security strength of at least 128 bits. ISO 29192 indicates that there are some lightweight cryptographic applications that may allow lower security requirements, that is, they do not have to assume powerful adversaries. In cases where 80-bit keys are used, this implies that less data can be encrypted safely with a single key before rekeying is required. It is therefore important that designers of cryptographic security systems make sure that the safe operation limitations of lightweight cryptographic mechanisms are not exceeded for a single key.

In 2018, NIST announced a project to solicit designs for lightweight cryptographic algorithms [NIST18]. NIST is planning to develop and maintain a portfolio of lightweight algorithms and modes that are approved for limited use. Each algorithm in the portfolio will be tied to one or more profiles, which consist of algorithm goals and acceptable ranges for metrics. NISTIR 8114 (*Report on Lightweight Cryptography*, March 2017) indicates that the initial focus is the development of authenticated encryption with additional data (AEAD) and secure hash functions. NIST has issued a preliminary set of two profiles for these algorithms [NIST17], one for implementations in both hardware and software and one for hardware-only implementations (Figure 14.2). The details of these profiles are shown in Tables 14.2 and 14.3. Note that the minimum security requirement is 112 bits.

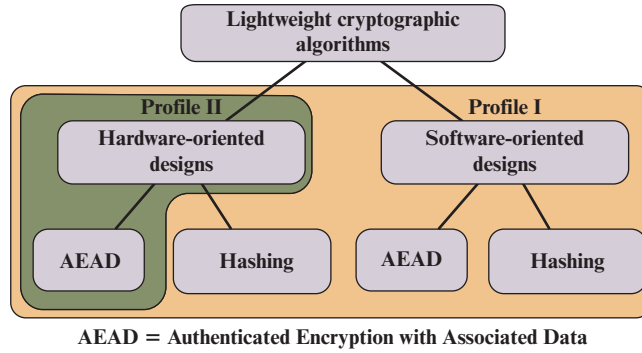


Figure 14.2 Profiles for Lightweight Cryptography

Table 14.2 Profile 1: AEAD and Hashing for Constrained Environments

Functionality	Authenticated Encryption with Associated Data and Hashing
Design goals	—Performs significantly better in constrained environments (hardware and embedded software platforms) compared to current NIST standards. —Both algorithms should be optimized to be efficient for short messages (e.g., as short as 8 bytes). —Message length shall be an integer number of bytes.
Physical characteristics	—Compact hardware implementations and embedded software implementations with low RAM and ROM usage should be possible.
Performance characteristics	—Performance on ASIC and FPGA should consider various standard cell libraries, the flexibility to support various implementation strategies (low energy, low power, low latency), with significant improvements over current NIST standards. —Performance on microcontrollers should consider a wide range of 8-bit, 16-bit, and 32-bit microcontroller architectures. —Preprocessing of a key (in terms of computation time and memory footprint) should be efficient.
Security characteristics	**AEAD** —A key length of 128 bits shall be supported. A longer key length may be supported, for example to provide security in the multi-key setting, or security against quantum computers. —Nonce lengths of up to 128 bits shall be supported. —Tag lengths of up to 128 bits shall be supported. —Plaintext lengths of up to $2^{50}-1$ bytes shall be supported. —Associated data of up to $2^{50}-1$ bytes shall be supported. —At least $2^{50}-1$ bytes can be processed securely under a single key. —Cryptanalytic attacks should require at least 2^{112} computations on a classical computer in a single-key setting. —Lends itself to countermeasures against various side-channel attacks, including timing attacks, simple and differential power analysis (SPA/DPA), and simple and differential electromagnetic analysis (SEMA/DEMA). **Hashing** —Cryptanalytic attacks should require at least 2^{112} computations on a classical computer. —Hash outputs of 256 bits must be supported, and longer hash values may be supported as well. —A maximum message length of $2^{50}-1$ bytes shall be supported. —Lends itself to countermeasures against various side-channel attacks, including timing attacks, simple and differential power analysis (SPA/DPA), and simple and differential electromagnetic analysis (SEMA/DEMA).

Side-Channel Attack Both ISO 29192 and NIST highlight the need for resistance to side-channel attacks. A side-channel attack is an attack enabled by leakage of information from a physical cryptosystem [TIRI07]. An attacker exploits the physical environment to recover some leakage that can be used to break the cryptographic algorithm. Characteristics that could be exploited in a

Table 14.3 Profile 2: AEAD for Constrained Hardware Environments

Functionality	Authenticated Encryption with Associated Data
Design goals	—Performs significantly better compared to current NIST standards. —Performance for short messages (e.g., as short as 8 bytes) is important. —Message length shall be an integer number of bytes.
Physical characteristics	—Targeted toward constrained hardware platforms. —Compact hardware implementations should be possible.
Performance characteristics	—Performance on ASIC and FPGA should consider a wide range of standard cell libraries and vendors. —Flexibility to support various implementation strategies (low energy, low power, low latency). —Preprocessing of a key (in terms of computation time and memory footprint) should be efficient.
Security characteristics	—A key length of 128 bits shall be supported. A longer key length may be supported, for example, to provide security in the multi-key setting, or security against quantum computers. —Nonce lengths of up to 128 bits shall be supported. —Tag lengths of up to 128 bits shall be supported. —Plaintext lengths of up to $2^{50} - 1$ bytes shall be supported. —Associated data of up to $2^{50} - 1$ bytes shall be supported. —At least $2^{50} - 1$ bytes can be processed securely under a single key. —Cryptanalytic attacks should require at least 2^{112} computations on a classical computer in a single-key setting. —Lends itself to countermeasures against various side-channel attacks, including timing attacks, simple and differential power analysis (SPA/DPA), and simple and differential electromagnetic analysis (SEMA/DEMA).

side-channel attack include running time, power consumption, and electromagnetic and acoustic emissions.

Figure 14.3 illustrates the basic operation of a side-channel attack. The attacker has access to the side-channel information emanating from the device, and may have either plaintext or ciphertext or both available. If operation is observable over an extended period of time, quite effective attacks are possible. The analysis consists in guessing key bits based on differences in the side-channel information. For example, the processing required for a 1 bit may be more than required for a 0 bit, and this affects processing time and power consumption. An attack on AES typically estimated the leakage caused by a single key byte. The result is that the entire 128-bit key can be found with 16×2^8 tests [TIRI07].

Constrained devices are often particularly vulnerable to side-channel attacks because they are located in environments that are not physically secure.

Countermeasures to side-channel attacks seek to eliminate, or at least diminish, the correlation between bits of the key and side-channel information. Examples of countermeasures include adding random delay to computations, inserting instruction cycles that have no effect in such a way that every cryptographic computation takes the same amount of time, and adding hardware logic that results in random amounts of power consumption.

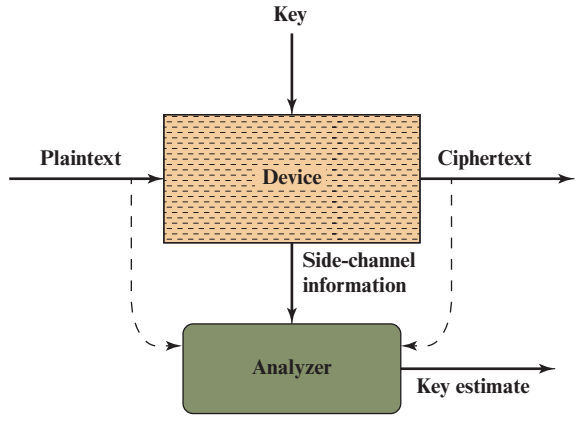


Figure 14.3 Side-Channel Attack

14.2 LIGHTWEIGHT CRYPTOGRAPHIC ALGORITHMS

To meet the requirements of lightweight cryptography, a number of new algorithms have been proposed [BIRY17, CRYP17]. Typical characteristics include:

- Many iterations of simple rounds
- Simple operations like XORs, rotation, 4 × 4 S-boxes, and bit permutations
- Smaller block sizes (e.g., 64 or 80 bits)
- Smaller key sizes (e.g., 96 or 112 bits)
- Simpler key schedules
- Small security margins by design
- Many iterations of simple rounds
- Simplified key schedules that can generate sub-keys on the fly

These design choices yield smaller security margins compared to established algorithms such as AES and SHA-2.

Authenticated Encryption with Additional Data

ARCHITECTURE STRATEGIES For both block and stream ciphers, the implementation to meet design goals makes use of one of three major hardware architecture options: parallel (loop unrolled), round-wise (rolled), and serial. Figure 14.4, based on one in [CRYP17], illustrates these options in general terms. A parallel implementation uses additional logic so that several round operations are performed in parallel. Typically, some form of pipelining is used so that during a given clock cycle, multiple rounds are being executed. In a round-wise, or rolled implementation, each round is executed separately, with execution of one round completed before the next round is begun. In both rolled and unrolled implementations, the architecture stores the full internal state, plus the key state if any, and then performs one round using a circuit operating on the full state at once. To achieve minimum chip area, a serial implementation can be used. With serial

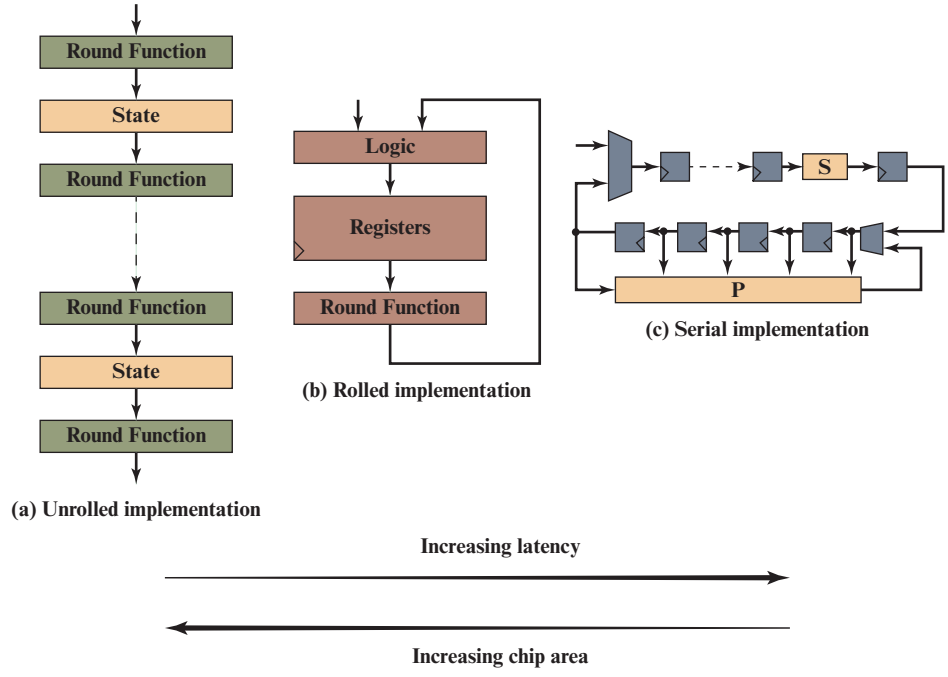


Figure 14.4 Basic Implementation Methods for Symmetric Ciphers

implementation, a block is processed in fractions, so that multiple operations are needed to complete a single round. With serial implementation, only a fraction of the state is updated at a time. As Figure 14.4 illustrates, moving from unrolled to rolled to serial implementation reduces the chip area required at the cost of increased execution time.

Block Ciphers Block ciphers are employed as the basic functional unit in a mode of operation to achieve encryption and in some authentication modes. Thus, they are intended for use processing multiple blocks of data. ISO 29192-1 indicates that the security of most modes of operation for block ciphers (including MAC and hash constructions) degrades at $q^2/2^n$, where n is the block size in bits and q is the number of blocks encrypted. For example, when $n = 64$, encryption of 2^{32} blocks is sufficient to expose the block cipher to attack. Therefore, care has to be taken since a shorter block size implies that less data can be encrypted using a single key.

An example of a lightweight cryptographic block cipher is the Scalable Encryption Algorithm (SEA) [STAN06]. SEA uses the Feistel cipher structure (Figure 4.3). SEA can have an arbitrary block size n (as long as $n = 6b$ for some b), word size, and number of rounds. It is based on the following operations:

- Bitwise exclusive-OR: $\oplus$
- Application of an S-box: S
- Rotation of the words in a vector of words: R = rotate left: R^{-1} = rotate right
- Bit rotation inside a word: r
- Addition modulo 2^b: $\boxplus$

The basic parameters are:

n: block size and key size

b: word size

$n_b = \frac{n}{2b}$: number of words per Feistel branch

n_r: number of rounds

The only constraint is that the block size must be a multiple of six times the word size (n is a multiple of $6b$). Thus, for an 8-bit processor, the block size can be 48, 96, 144, and so on.

Figure 14.5 illustrates the functionality of a single round. For each round, a block is divided into left and right halves and the round operations are:

Encryption	$L_{i+1} = R_i$ $R_{i+1} = \mathrm{R}(L_i) \oplus \mathrm{r}(\mathrm{S}(R_i \boxplus K_i))$
Decryption	$L_{i+1} = R_i$ $R_{i+1} = \mathrm{R}^{-1}(L_i \oplus \mathrm{r}(\mathrm{S}(R_i \boxplus K_i)))$
Subkey generation	$KL_{i+1} = KR_i$ $KR_{i+1} = KL_i \oplus \mathrm{R}(\mathrm{r}(\mathrm{S}(KR_i \boxplus C_i)))$

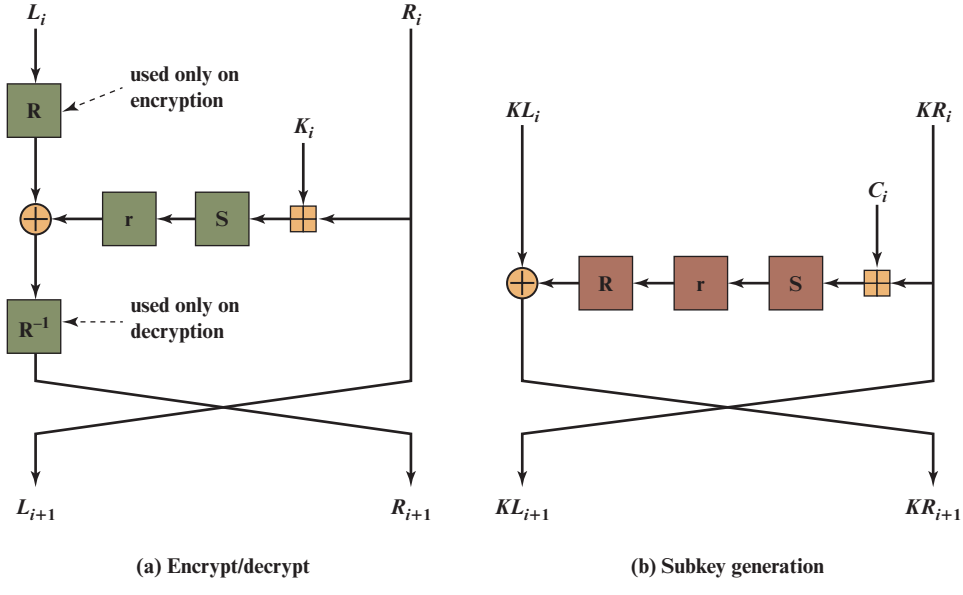


Figure 14.5 One Round of Scalable Encryption Algorithm

The substitution box is defined by a 3-bit substitution table. For a 3-bit chunk x:

x	000	001	010	011	100	101	110	111
$S(x)$	000	101	110	111	100	011	001	010

Data can be processed in blocks of 3 words (24 bits) at a time, providing opportunity for parallel implementation of the S-box substitution to the eight 3-bit chunks.

The constant C_i is a n_b-word vector in which all of the words have the value 0 except the least significant word, which has the value i.

SEA has a number of strengths for use in a constrained device. Only a few operations need to be implemented. It is easily scalable in terms of both block and key size. SEA is designed to provide good nonlinearity and diffusion. The authors look at various types of attacks to justify the design decisions in the creation of SEA [STAN06]. A number of studies have shown that SEA provides a good balance of compact implementation and performance [KUMA11a, KUMA10 CAKI10, MACE08].

Stream Ciphers Stream ciphers are also a promising approach to symmetric encryption for constrained environments. Chapter 8 presents one example of a stream cipher that is suitable for constrained devices: Grain-128.

Hash Functions

Traditional hash functions may not meet the requirements for implementation on constrained devices. NISTIR 8114 points out two ways in which lightweight hash functions differ from more traditional ones:

- ***Smaller internal state and output sizes:*** Large output sizes are important for applications that require collision resistance of hash functions. For applications that do not require collision resistance, smaller internal states and output sizes might be used. When a collision-resistant hash function is required, it may be acceptable that this hash function has the same security against preimage, second-preimage, and collision attacks. This may reduce the size of the internal state.
- ***Smaller message size:*** Conventional hash functions are expected to support inputs with very large sizes (around 2^{64} bits). In most of the target protocols for lightweight hash functions, typical input sizes are much smaller (e.g., at most 256 bits). Hash functions that are optimized for short messages may therefore be more suitable for lightweight applications.

An example of a lightweight cryptographic hash functions is PHOTON [GUO11]. PHOTON is one of the hash functions specified in ISO 29192. It is also listed in [CRYP17].

PHOTON uses a sponge structure, similar to that used by SHA-3, as shown in Figure 14.6. Sponge functions have been well studied in terms of security and can be designed for compact implementation. The sponge function has three main elements:

- An internal state of t bits consisting of a c-bit capacity and an r-bit rate $(t = c + r)$. The rate r is the number of bits processed at each iteration, and the capacity c is a measure of the complexity of the construction and therefore its security. The hash size n is equal to c.

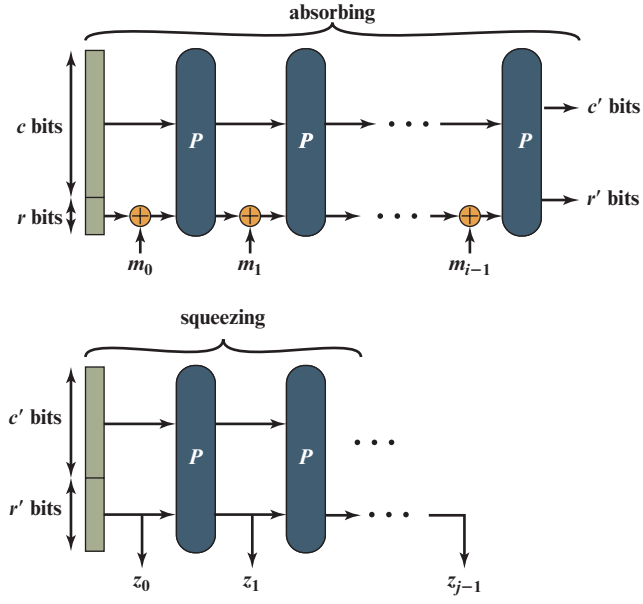


Figure 14.6 Photon Sponge Structure

- A permutation function P that operates on the internal state at each iteration.
- A padding function that appends sufficient bits to the data input.

The sponge structure consists of an absorbing phase that absorbs the message blocks into an internal state, followed by a squeezing phase that generates the hash blocks. For the absorbing phase, the data input, or message, is padded by appending a 1 bit and as many zeros as needed so that the input length is an integral multiple of r. The input is divided into i r-bit message blocks $m_0, \ldots, m_{i-1}$. The internal state is initialized to the value $S_0 = IV = \{0\}^{t-24}||n/4||r||r'$, where the three values are each coded in 8 bits and n is the hash size. For each of the i iterations, m_i is XORed with the rate portion of the internal state and then the permutation P is applied to the t-bit state.

For the squeezing phase, the internal state is divided into r' and c' sections, which may differ in lengths from r and c. Increasing r' reduces the time spent in the squeezing phase but might reduce preimage security. This phase produces a sequence of ir'-bit hash blocks $z_0, \ldots, z_{j-1}$, with $j = \lceil n/r' \rceil - 1$. The hash output is $z_0|| \ldots ||z_{j-1}$. If the hash output is not a multiple of r', it is truncated to n bits.

Using the structure of Figure 14.6, five variants of PHOTON are defined, as shown in Table 14.4. The five versions provide increasing levels of security at the cost of increasing size and processing time. Note the small size of the internal state, of between 100 and 288 bits. By contrast, SHA-3 has an internal state of 1600 bits, and SHA-512 has an internal state of 512 bits.

The internal structure of the permutation function consists of unkeyed AES-like primitives especially derived for hardware optimization. The advantage of AES-like primitives is that PHOTON takes advantage of the previous cryptanalysis

Table 14.4 PHOTON Versions

PHOTON-$n/r/r'$	n (Hash Size)	r (Message Block Size)	r' (Hash Block Size)	t (Internal State)
PHOTON-80/20/16	80	20	16	100
PHOTON-128/16/16	128	16	16	144
PHOTON-160/36/36	160	36	36	196
PHOTON-224/32/32	224	32	32	256
PHOTON-256/32/32	256	32	32	288

Note: All values are expressed in bits.

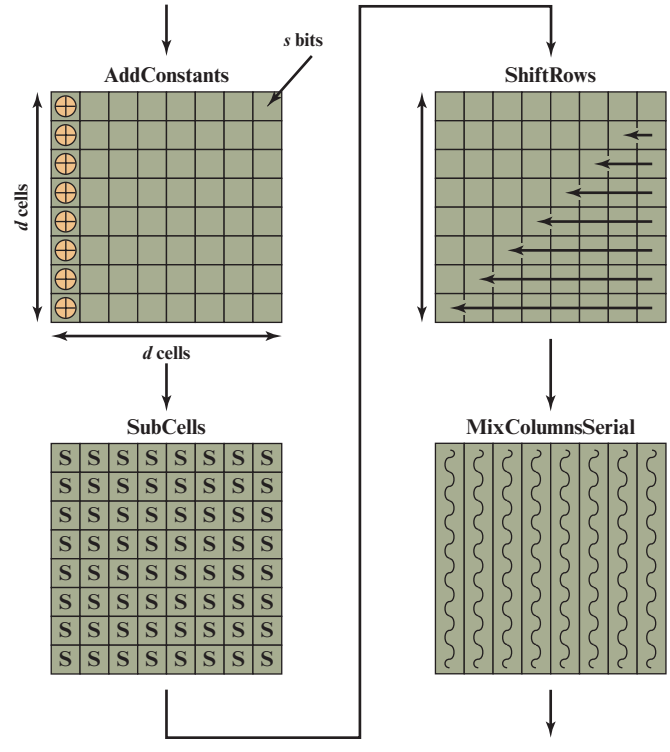


Figure 14.7 One Round of a PHOTON Permutation

performed on AES and on AES-based hash functions. Figure 14.7 illustrates the permutation P structure. The t-bit internal state is organized of a matrix of $(d \times d)$ s-bit cells. Thus, $i = (d \times d) + s$. The permutation consists of 12 rounds of four stages:

- **AddConstants:** Round constants are XORed to the first column of the matrix.
- **SubCells:** An S-box is used to map each matrix entry to a new value.
- **ShiftRows:** The position of the cells in each row is rotated, as illustrated.
- **MixColumnsSerial:** This function linearly mixes all the columns independently.

The authors claim that PHOTON is extremely lightweight, very close to the theoretical optimum and achieves excellent area/throughput trade-offs.

Message Authentication Codes

[CRYP17] points out that there are two approaches to developing a lightweight message authentication code (MAC). The first approach is to use an existing MAC with an underlying lightweight cryptographic algorithm. The most prominent examples are CMAC and HMAC, both discussed in Chapter 12. Because the overheads of CMAC and HMAC are not high, a lightweight MAC can be implemented by configuring these algorithms with an underlying lightweight cryptographic algorithm. In the case of CMAC, this means using a lightweight symmetric encryption algorithm. In the case of HMAC, this means using a lightweight hash algorithm.

The second approach is to specifically design a new lightweight MAC algorithm. There has been much more work done on lightweight encryption algorithms and cryptographic hash codes than on lightweight MAC algorithms.

One example of a newly designed MAC is SipHash [AUMA12]. It is the only MAC listed in [CRYP17] and it has been widely implemented. The principal objectives for the design of SipHash were:

- Optimize the MAC algorithm for short messages. This is in keeping with the typical exchanges be constrained devices.
- Build a MAC that is secure, efficient, and simple.

Two important characteristics of SipHash are that it does not require key expansion and that minimal internal state is required. SipHash has a sponge type of structure consisting of a compression phase, during which the message is absorbed and compressed, followed by a finalization phase, which provide further mixing of the bits. SipHash is a family of functions denoted SipHash-c-d, where c is the number of compression rounds between message blocks and d is the number of finalization rounds. The rounds, denoted SipRound, are identical for the two phases. The variables used in all of the SipHash variants are as follows:

- A 128-bit key k, divided into two 64-bit blocks k_0 and k_1.
- A b-byte message m, that is divided into $w = \lceil (b + 1)/8 \rceil$ 64-bit blocks $m_0, \ldots, m_{w-1}$, where m_{w-1} includes the last (b mod 8) bytes of m followed by null bytes ending with a byte encoding the positive integer b mod 256.
- An internal state consisting of four 64-bit words, labeled v_0, v_1, v_1, v_2.
- A 64-bit tag. This is the output of the SipHash function, used for message authentication.

Figure 14.8a illustrates the compression state. To begin, the internal state is initialized as:

$$v_0 = k_0 \oplus \mathrm{C}_0 = k_0 \oplus 736\mathrm{f}6\mathrm{d}6570736575$$
$$v_1 = k_1 \oplus \mathrm{C}_1 = k_1 \oplus 646\mathrm{f}72616\mathrm{e}646\mathrm{f}6\mathrm{d}$$
$$v_2 = k_0 \oplus \mathrm{C}_2 = k_0 \oplus 6\mathrm{c}7967656\mathrm{e}657261$$
$$v_3 = k_1 \oplus \mathrm{C}_3 = k_1 \oplus 7465646279746573$$

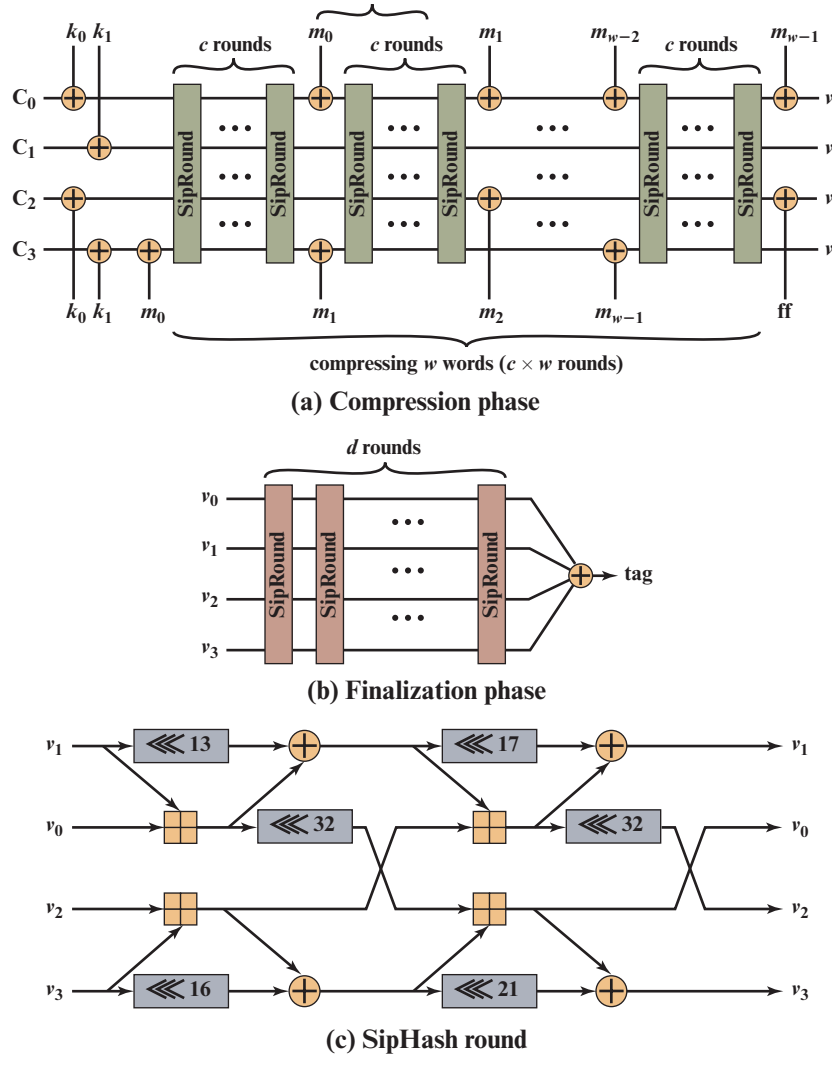


Figure 14.8 SipHash Message Authentication Code

Thus, each half of each key is subject to two different bit flipping operations. Then each message word is processed as follows:

1. The internal state is modified by $v_3 = v_3 \oplus m_0$.
2. This is followed by c iterations of SipRound.
3. This is followed by $(w - 1)$ steps consisting of

   $v_0 = v_0 \oplus m_{i-1}$

   $v_3 = v_3 \oplus m_i$

   c iterations of SipRound

4. This is followed by

$$v_0 = v_0 \oplus m_{w-1}$$
$$v_3 = v_3 \oplus \text{ff}$$

In the finalization phase (Figure 14.8b), there are d more rounds applied to the internal state. Then, the 64-bit tag is generated as $v_0 \oplus v_1 \oplus v_2 \oplus v_3$.

The function SipRound transforms the internal state using the simple functions of addition, exclusive-OR, and bitwise left logical rotate, as shown in Figure 14.8c.

The authors believe that SipHash-2-4 provides strong security and is the recommended "fast" option. SipHash-4-8 is a conservative choice, providing higher security at about half the speed.

Asymmetric Cryptographic Algorithms

Neither NISTIR 8114 nor [CRYP17] mention developing lightweight asymmetric cryptographic algorithms. So far, there has been little interest in this area. Asymmetric algorithms typically operate on only small blocks of data and are relatively infrequently invoked. Thus, there is less motivation for attempting lightweight versions. Additionally, most asymmetric algorithms are already relatively compact.

One potential application is to use a lightweight hash function in a digital signature algorithm.

14.3 POST-QUANTUM CRYPTOGRAPHY CONCEPTS

Post-quantum cryptography is concerned with the development of cryptographic algorithms that are secure against the potential development of quantum computers. Whereas lightweight cryptography is primarily concerned with the efficiency and compactness of symmetric encryption algorithms and cryptographic hash functions, post-quantum cryptography is concerned with the security of asymmetric cryptographic algorithms.

We begin with a brief introduction to quantum computing and then look at the implications for asymmetric cryptography.

Quantum Computing

Quantum computing is based on the representation of information in a form analogous to the behavior of elementary particles in quantum physics. A practical application of this representation, in terms of performing calculations, requires producing a physical system that performs computation making use of quantum physical principles. As yet, no such general-purpose computing system has been developed but in principle it is possible to do so.

Information in a quantum computer is represented as quantum bits, or qubits. A qubit can be viewed as a quantum analog of a classical bit, one that obeys the laws of quantum physics. In particular, qubits have two properties that are relevant to quantum computing:

- Superposition: A qubit does not exist in a single state but in a superposition of different states. It is only when a measurement is taken that the qubit collapses into a unique state (binary 1 or 0). Prior to that it is only possible to express a

probability that the qubit is a 1 or a 0. The qubit can be thought of a vector of unit magnitude in a two-dimensional vector space.

- Entanglement: Qubits can be linked to each other over the course of operations reflecting the physical phenomenon known as quantum entanglement. The relevant implication of this is that state of a multiple-qubit system is not represented by a linear combination of the state vectors of each qubit but rather a tensor product.

It is well beyond the scope of this brief introduction to explain the implications of these two properties in terms of computation. In essence, because of entanglement, a set of multiple qubits has a state space that grows exponentially with the number of qubits. Because of the superposition of states, one operator applied to the set operates on all the states in parallel. This enables allowing quantum computers to look through millions of potential solutions at once, rather than sequentially. Thus, computational power scales exponentially.

The challenges of building a practical quantum computer are immense. The various physical realizations of qubits that are being investigated are very fragile, with some requiring extremely cold temperatures. As reported in [GREE18], quantum computing systems will need new algorithms, software, interconnects, and a number of other yet-to-be-invented technologies specifically designed to take advantage of system's tremendous processing power—as well as allow the computer's results to be shared or stored.

Shor's Factoring Algorithm

Public-key cryptography supports three critical cryptographic functionalities: public-key encryption, or asymmetric encryption, digital signatures, and key exchange. The underlying algorithms that are primarily implemented for these functions are Diffie-Hellman key exchange, the RSA cryptosystem, and elliptic curve cryptosystem. In turn, the security of these algorithms depends on the difficulty of solving certain number theoretic problems, mainly integer factorization or discrete logarithms.

Shor [SHOR97] has described algorithms designed for a quantum computer (operate on qubits) for prime factorization and discrete logarithms that execute in polynomial time. For example, the number of steps in the factorization algorithm grows polynomial to the number of digits of the integer to be factored. The implication of Shor's work is profound for public-key systems. For example, a white paper from the European Telecommunications Standards Institute [ETSI14] indicates that to attack a 3072-bit RSA key, a quantum computer must have a few thousand logical qubits. If and when quantum computers that can handle that number of qubits is practical, such a key is no longer safe. Further, using Shor's algorithm, the number of qubits needed scales linearly with the bit length of the RSA or ECC key. Moving to a larger RSA key provides security only until a larger quantum computer is built. And, as [ETSI14] points out, doubling the size of an RSA or ECC key doubles the burden on a quantum computer, but increases the running time for using the keys on a conventional computer by a factor of 8. This type of response to quantum computing is clearly unsustainable.

Figure 14.9 illustrates the impact for RSA. The individual diamonds indicate the year when a given RSA key length was demonstrated to be broken. The progress is due to a combination of increased computing power and more sophisticated cryptanalytic algorithms. Based on the trend line, a key size of 1024 bits is secure for the near future and a key size of 2048 bits is secure for a very long time. However, if practical

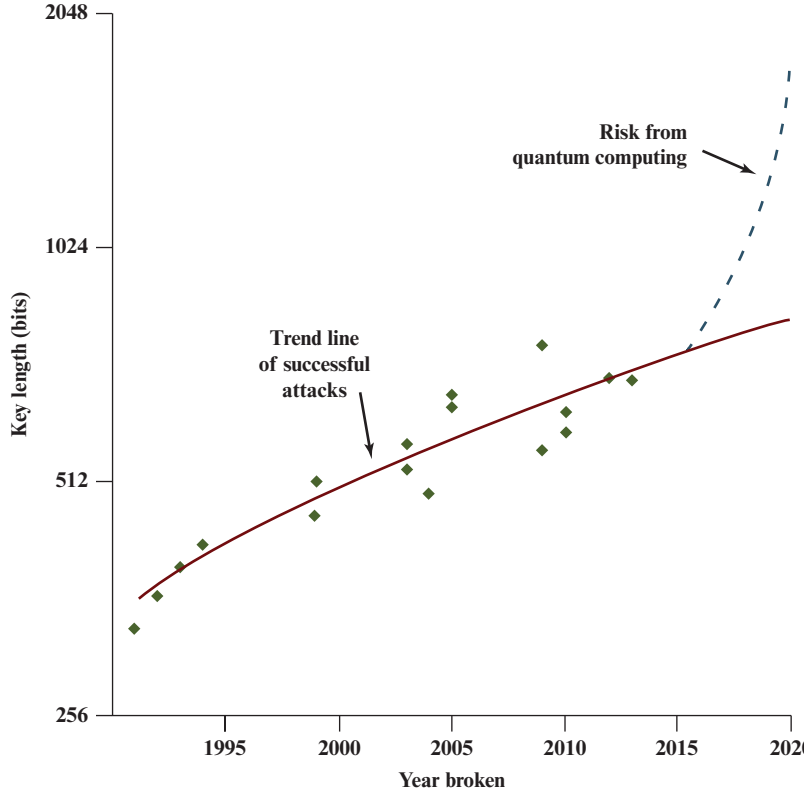


Figure 14.9 RSA Key Lengths Broken by Conventional Computing Architectures

quantum computers are introduced, and Shor's algorithm is used, the trend line could become exponential, the key length of 2018 could be broken relatively soon.

Grover's Algorithm

Grover's algorithm [GROV96] searches an unordered list in $O(\sqrt{n})$ time, while conventional algorithms require $O(n)$. This is not as dramatic as the speedup achieved by Shor's algorithms, but it is a significant improvement for the type of brute-force approach typically used for symmetric encryption and hash algorithms. Grover's algorithm can reduce the cost of attacking a symmetric cryptographic algorithm. For a cryptographic algorithm with a key size of n bits, Grover's algorithm can theoretically reduce the security of that algorithm to one with a key size of $n/2$ bits. This is not nearly as serious as the threat to asymmetric algorithms posed by Shor's algorithm. For example, a 128-bit AES key is considered secure for the foreseeable future. To guard against a quantum attack using Grover's algorithm, the same level of security could be maintained by moving to a 256-bit key. Similarly, Grover's algorithm can theoretically reduce the security of a cryptographic hash algorithm by a factor of two. This can be countered by doubling the hash length.

Furthermore, it has been shown that an exponential speed up for search algorithms is impossible, suggesting that existing symmetric algorithms and hash functions should be secure in a quantum era [BENN97].

Cryptoperiods

Although practical large-scale quantum computers are not likely for a number or years, there has been considerable interest and some urgency in developing cryptographic algorithms that are secure against such computers. The following are examples:

- In 2015, the U.S. National Security Agency (NSA) released a major policy statement on the need for post-quantum cryptography. Prior to this, NSA had defined a suite of algorithms (Suite B) that were approved for protection of both sensitive but unclassified (SBU) and classified information, including the approval of ECC. The 2015 NSA statement indicated that partners and vendors that had not yet implemented Suite B should not expend additional resources on developing ECC products because NSA planned to transition to post-quantum algorithms in the foreseeable future.
- In 2016, NIST announced a request for submissions for public-key **post-quantum cryptographic algorithms**. Round 2 submissions have been received and, as of this writing, are being evaluated.
- In 2014, the ETSI Quantum Safe Cryptography (QSC) Industry Specification Group was formed to assess and make recommendations for quantum-safe cryptographic primitives and protocols.

To understand the motivation for rapid progress in this area, we need to discuss the concept of **cryptoperiod**. The cryptoperiod of a cryptographic key is the time span during which a specific cryptographic key is authorized for use for its defined purpose. This is an important consideration. A number of potential security threats make it advisable that any key not be used for a prolonged period of time. These threats include:

- **Brute-force attacks:** As raw processing power and the ability to use numerous processors in parallel increase, a given key length becomes increasingly vulnerable and longer key lengths are advised. Any of the shorter keys in use need to be retired as quickly as possible and longer key lengths employed. For example, NIST used to recommend the use of 1024-bit keys for certain asymmetric algorithms but now recommends 2048 bits for these algorithms.
- **Cryptanalysis:** Over time, flaws may be discovered in a cryptographic algorithm that make it feasible to "break" the algorithm. An example of this is the original NIST standard hash algorithm, SHA-1, which was used in their Digital Signature Algorithm. Once these weaknesses were discovered, NIST migrated to SHA-2 and SHA-3. Similarly, methods have been found for breaking algorithms such as the RSA asymmetric algorithm at rates faster than brute force, which can be thwarted by using longer keys.
- **Other security threats:** Beyond simply attacking an algorithm directly in an attempt to discover a key that is being used, there are a variety of other methods of attack. This includes attacks on the mechanisms and protocols associated with the keys, key modification, and achieving unauthorized disclosure. The longer a particular key is used for encryption and decryption, the greater the chance that some means of learning the key will succeed.

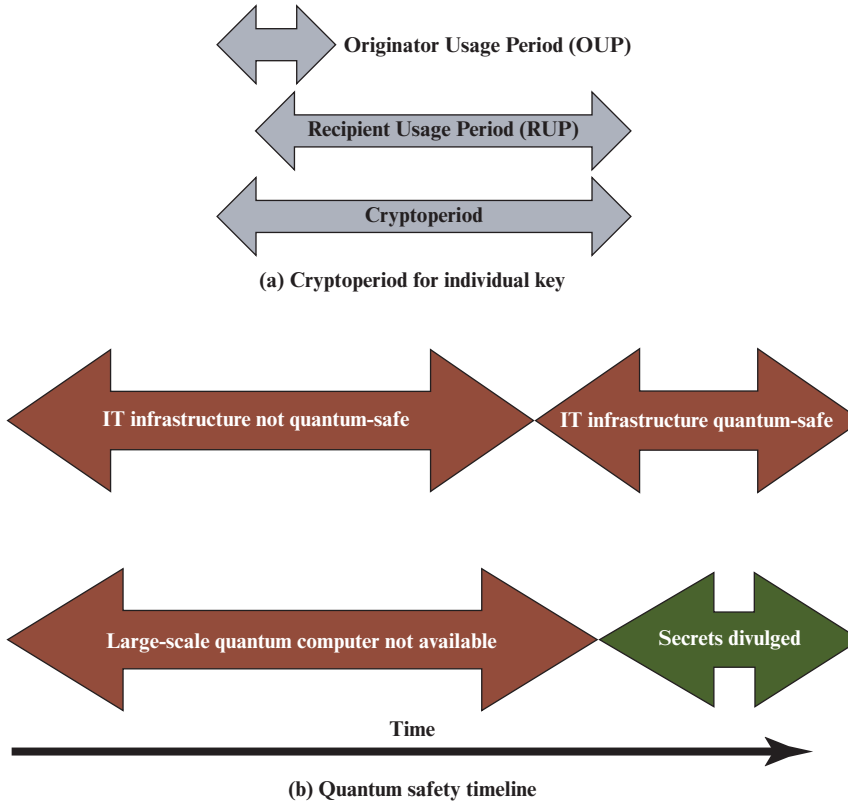


Figure 14.10 Lead Time for Quantum Safety

Accordingly, an enterprise should have policies for the maximum cryptoperiod of each key type.

Figure 14.10a illustrates the two aspects of a cryptoperiod. The originator usage period (OUP) refers to the time during which data may be encrypted, and the recipient usage period (RUP) is the time during which such data may continue to be maintained in its encrypted form and subject to decryption. The RUP often starts at the beginning of the OUP, but there may be some delay before data can be decrypted. More significantly, the end of the RUP may extend for a considerable length of time beyond the end of the OUP. That is, the policy may state that a given key may no longer be used for encrypting new data, but the data that have already been encrypted may be retained in the encrypted form, available for decryption for a further period of time. Hence the cryptoperiod extends from the start of the OUP to the end of the SUP. Table 14.5 shows the cryptoperiods suggested in SP 80-57.

Quantum Safety

Equivalent to the term post-quantum cryptography is the term *quantum-safe cryptography*. The latter term emphasizes the need for creating cryptographic algorithms that are safe, or secure, against quantum computing algorithms. Figure 14.10b illustrates what this means in terms of times. At present, no organization

Table 14.5 Suggested Cryptoperiods from SP 800-57

Key Type	OUP	RUP
1. Private Signature Key	1 to 3 years	—
2. Public Signature-Verification Key	Several years (depends on key size)	
3. Symmetric Authentication Key	≤ 2 years	≤ OUP + 3 years
4. Private Authentication Key	1 to 2 years	
5. Public Authentication Key	1 to 2 years	
6. Symmetric Data Encryption Keys	≤ 2 years	≤ OUP + 3 years
7. Symmetric Key Wrapping Key	≤ 2 years	≤ OUP + 3 years
8. Symmetric RBG Keys	See [SP800-90]	—
9. Symmetric Master Key	About 1 year	—
10. Private Key Transport Key	≤ 2 years	
11. Public Key Transport Key	1 to 2 years	
12. Symmetric Key Agreement Key	1 to 2 years	
13. Private Static Key Agreement Key	1 to 2 years	
14. Public Static Key Agreement Key	1 to 2 years	
15. Private Ephemeral Key Agreement Key	One key-agreement transaction	
16. Public Ephemeral Key Agreement Key	One key-agreement transaction	
17. Symmetric Authentication Key	≤ 2 years	
18. Private Authentication Key	≤ 2 years	
19. Public Authentication Key	≤ 2 years	

or IT installation is using post-quantum cryptographic algorithms and so cannot be considered quantum safe. This situation is satisfactory until such time as large scale-quantum computers are available. If such computers become available prior to the widespread introduction of post-quantum algorithms, then there will be a period of time in which all IT installations are vulnerable to attack. Thus, there is some urgency in developing and deploying post-quantum algorithms.

The issue of timing also relates to the concept of the cryptoperiod. Any IT installation managing a large number of symmetric and asymmetric keys with different end dates for the respective cryptoperiods. The aggregate of all those keys and their cryptoperiods indicate how long it is after post-quantum cryptography is introduced before all pre-quantum keys are phased out.

As pointed out in [ETSI14], and illustrated in Figure 14.11, three levels of security-related entities are vulnerable to quantum attack:

- **Cryptosystems:** A cryptosystem consists of a set of cryptographic algorithms together with the key management processes that support use of the algorithms in some application context. Any cryptosystem that relies on the security of integer factoring or discrete logarithms is vulnerable. This includes RSA, DSA, DH, ECDH, ECDSA, and other variants of these ciphers.

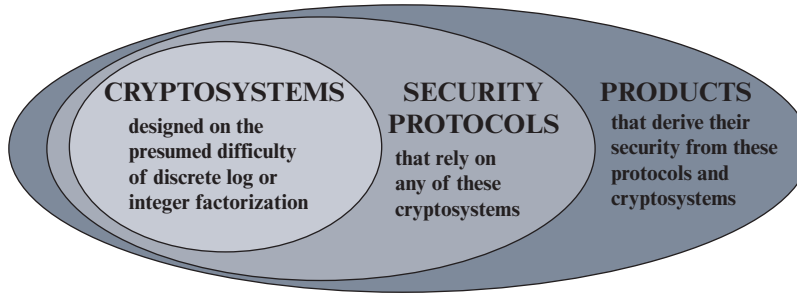


Figure 14.11 Entities Vulnerable to Quantum Computing

Table 14.6 Impact of Quantum Computing on Common Cryptographic Algorithms

Cryptographic Algorithm	Type	Purpose	Impact from Large-Scale Quantum Computer
AES	Symmetric key	Encryption	Larger key sizes needed
SHA-2, SHA-3	Cryptographic hash	Hash function	Larger output needed
RSA	Asymmetric key	Signature, key establishment	No longer secure
ECDSA, ECDH (elliptic curve cryptography)	Asymmetric key	Signature, key exchange	No longer secure
DSA (finite field cryptography)	Asymmetric key	Signature, key exchange	No longer secure

Almost all public key cryptography in fielded security products and protocols today use these types of ciphers.

- **Security protocols or security components of network protocols:** Any such protocols that derive security from the public-key algorithms listed in the preceding bullet are vulnerable.
- **Products:** Any products or security systems that derive security from the above protocols are vulnerable.

Cryptosystems or portions of cryptosystems that employ symmetric ciphers or hash functions can be made quantum safe by increasing the size of the key or the hash length, respectively. It is public-key systems that are of concern. Table 14.6 summarizes these considerations.

14.4 POST-QUANTUM CRYPTOGRAPHIC ALGORITHMS

The types of asymmetric algorithms that are vulnerable to quantum computing are in the following categories:

- **Digital signatures:** Public-key signature algorithms for generating and verifying digital signatures.

Table 14.7 Submissions to NIST Post-Quantum Cryptography Competition

	Signatures	KEM/Encryption	Total
Lattice-based	4	24	28
Code-based	5	19	24
Multivariate	7	6	13
Hash-based	4	—	4
Other	3	10	13
Total	23	59	82

- **Encryption:** Used for encrypting symmetric keys for transport from one party to another. Also used in various key establishment algorithms. In general terms, these proceed as follows: Each party has either one or two key pairs, and the public keys are made known to the other party. The key pairs are used to compute a shared secret value, which is then used with other information to derive keying material using a key derivation function.
- **Key-Establishment Mechanisms (KEMs):** Refers to schemes such as Diffie-Hellman key exchange.

There is no single widely accepted alternative to the existing algorithms based on integer factorization or discrete logarithms. Of the approaches reported in the literature, four general types of algorithms predominate:

- **Lattice-based cryptography:** These schemes involve the construction of primitives that involve lattices.
- **Code-based cryptography:** These schemes are based on error-correcting codes.
- **Multivariate polynomial cryptography:** These schemes are based on the difficulty of solving systems of multivariate polynomials over finite fields.
- **Hash-based signatures:** These are digital signatures constructed using hash functions.

An indication of the interest shown in these approaches is found in the submissions to the NIST effort at post-quantum standardization. As reported in NISTIR 8105 (*Report on Post-Quantum Cryptography*, April 2016), NIST hopes to standardize a number of algorithms that can be used to replace or complement existing asymmetric schemes. For the first round, NIST has received 82 submissions, broken down as shown in Table 14.7.

There are several reasons why NIST does not intend to settle on a single standard:

- The requirements for public-key encryption and digital signatures are more complicated than those of symmetric encryption and cryptographic hash functions.
- The current scientific understanding of the power of quantum computers is far from comprehensive.

- Some of the candidate post-quantum cryptosystems may have completely different design attributes and mathematical foundations, so that a direct comparison of candidates would be difficult or impossible.
- The various approaches exhibit different advantages and disadvantages, beyond considerations of security.

Although there will be significant differences within each of the four approaches listed above, the following general statements can be made:

- **Lattice-based cryptography:** These schemes are relatively simple, efficient, and highly parallelizable.
- **Code-based cryptography:** These schemes are quite fast but require very large key sizes.
- **Multivariate polynomial cryptography:** For digital signatures, these schemes require very large key sizes.
- **Hash-based signatures:** Many of the more efficient hash-based signature schemes have the drawback that the signer must keep a record of the exact number of previously signed messages, and any error in this record will result in insecurity. Another drawback is that they can produce only a limited number of signatures. The number of signatures can be increased, even to the point of being effectively unlimited, but this also increases the signature size.

Because of the complexity of the mathematics and the implementation of these types of schemes, a full description is beyond our scope. The remainder of this section provides a brief overview of each of the four approaches.

Lattice-Based Cryptographic Algorithms

An m-dimensional lattice of rank n is the set of vectors that can be expressed as the sum of integer multiples of a specific set of n vectors, collectively called the basis of the lattice. More formally, a lattice can be defined as:

$$L = \left\{ \sum_{i=1}^{n} x_i \boldsymbol{b}_i \mid n_i \in \mathbb{Z}, \boldsymbol{b}_i \in \mathbb{R}^m \right\}$$

where the $\boldsymbol{b_i}$s are linearly independent vectors of length m over the real numbers and the x_i are integers. The set of vectors $\boldsymbol{b_i}$ is called a lattice basis. The lattice basis can be represented by a matrix $\mathbf{B}$, where the ith column of the matrix is $\boldsymbol{b_i}$. We refer to m as the dimension of the lattice, and n the rank of the lattice. A lattice can be depicted as n points defined by the basis in m-dimensional space; that is, each point is the end point of one of the basis vectors. A lattice is said to be full-rank when $n = m$. There are infinitely many lattices of the same dimension.

A basis vector $\boldsymbol{b_i}$ consists of m real numbers $(b_{i,1}, \ldots, b_{i,m})$. The length of the vector is the real number:

$$\|\boldsymbol{b_i}\| = \sqrt{b_{i,1}^2 + b_{i,2}^2 + \cdots + b_{i,m}^2}$$

The basis for a given lattice is not unique. The existence of multiple bases for the same lattice is important for the development of cryptographic algorithms, because

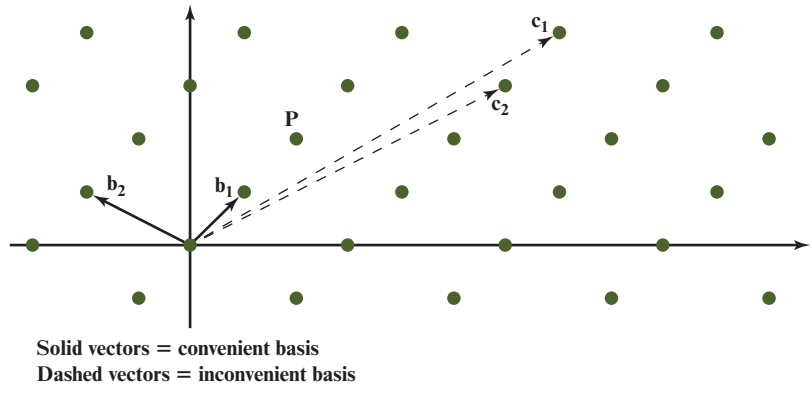


Figure 14.12 Two Bases for a Two-Dimensional Lattice

some bases are easier to handle than others. Figure 14.12, which depicts a lattice with $n = m = 2$, illustrates the concepts just introduced. The basis $\mathbf{b}_1 = (0.5, 0.5)$, $\mathbf{b}_2 = (-1, 0.5)$, with lengths $\|\boldsymbol{b}_1\| = 0.707$ and $\|\boldsymbol{b}_2\| = 1.12$, defines all of the points shown in the figure. For example, the point $\mathbf{P} = (1, 1)$ is equal to $x_1\mathbf{b}_1 + x_2\mathbf{b}_2 = x_1(0.5, 0.5) + x_2(-1, 0.5)$, for $x_1 = 2, x_2 = 0$. The same lattice is also defined by the basis $\mathbf{c}_1 = (3.5, 2), \mathbf{c}_2 = (3, 1.5)$, with lengths $\|\boldsymbol{c}_1\| = 4.03$ and $\|\boldsymbol{c}_2\| = 3.35$. For example, $\mathbf{P} = (1, 1)$ is equal to $x_1\mathbf{c}_1 + x_2\mathbf{c}_2$ for $x_1 = 2/3$, $x_2 = -2/3$. With either basis, any point in the space can be defined by a linear combination of its two vectors, but the basis $\mathbf{b}_1, \mathbf{b}_2$ is computationally more convenient.

The essence of a lattice-based cryptographic algorithms is to exploit a hard problem in lattices. One such problem is the Closest Vector Problem (CVP) which can be stated as follows: given a basis of a lattice L and a vector $v \in \mathbb{R}^m$, find a lattice vector that minimizes the distance to v. Note that, in general, v defines a point that is not part of the lattice. The Shortest Vector Problem (SVP) is to find the shortest nonzero vector within a lattice. There is no known quantum algorithm for solving CVP or SVP for lattices of large dimension. In practice, the cryptographic algorithms that have been proposed assume that a relaxed variant of CVP or SVP is still hard to solve.

The most widely studied lattice-based approach is the NTRU family of cryptographic algorithms. Such algorithms use a specific class of lattices that have an extra symmetry. In all NTRU-based schemes, the private key represents a lattice basis consisting of short vectors, while the public key represents a lattice basis consisting of longer vectors. In general terms, these algorithms work as follow. A message is encoded as a vector $\mathbf{m}$. A random point in the lattice defined by the private key basis is added to $\mathbf{m}$ to form a vector $\mathbf{e}$. The public key has been defined by multiplying the private key basis $\mathbf{B}$ by a matrix $\mathbf{U}$, yielding another basis $\mathbf{B'}$ for the same lattice, with longer vectors. To decrypt the message, find the lattice point closest to the ciphertext vector $\mathbf{C} = \mathbf{eU}^{-1}$, and subtract it from the ciphertext vector. The result is the original plaintext vector. That is, $\mathbf{B'}$ is the public key and it is possible to find the lattice point $\mathbf{X}$ closest to $\mathbf{C}$, such that $\mathbf{C} - \mathbf{X} = \mathbf{m}$. However, given $\mathbf{B'}$, it is computationally infeasible to determine $\mathbf{B}$.

The matrix **U** in the above scheme must be a unimodular matrix, which means that the determinant of **U** is 1 or −1. For example, the matrix

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

is unimodular with a determinant of 1. It can be shown that the inverse $\mathbf{U}^{-1}$ of unimodular matrix **U** is also unimodular, and that two bases $\mathbf{B_1}$, $\mathbf{B_2}$ define the same lattice if and only if $\mathbf{B_2} = \mathbf{B_1U}$.

A number of variations on this basic scheme are currently being pursued [LAUT17].

Code-Based Cryptographic Algorithms

An error correction code (ECC) allows data that is being read or transmitted to be checked for errors and, when necessary, corrected. Figure 14.13 illustrates in general terms how the process is carried out. On the source end, each k-bit block of data is mapped into an n-bit block ($n > k$) called a codeword, using an ECC encoder.

The ECC is referred to as an (n, k) ECC. Encoding can be described as multiplying a k-bit data vector **m** by a $k \times n$ matrix **G** to yield an n-bit codeword vector **c**:

$$\mathbf{c} = \mathbf{mG}$$

For each generator matrix, there is an $(n - k) \times k$ parity check matrix **H** whose rows are orthogonal to those of **G**; that is, $\mathbf{GH^T} = \mathbf{0}$.

The codeword, whether stored or transmitted, is subject to impairments, which may produce one or more bit errors in the block. At the destination, the received codeword may contain errors. This block is passed through an ECC decoder, with one of four possible outcomes:

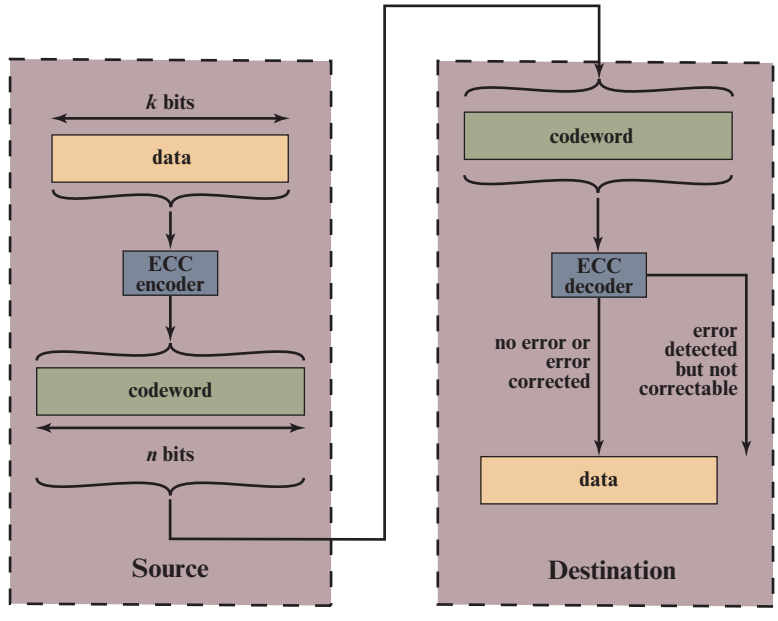


Figure 14.13 (n, k) Error Correction Code

- **No errors:** If there are no bit errors, the input to the ECC decoder is identical to the original codeword, and the decoder produces the original data block as output.
- **Detectable, correctable errors:** For certain error patterns, it is possible for the decoder to detect and correct those errors. Thus, even though the incoming data block differs from the transmitted codeword, the decoder is able to map this block into the original data block.
- **Detectable, not correctable errors:** For certain error patterns, the decoder can detect but not correct the errors. In this case, the decoder simply reports an uncorrectable error.
- **Undetectable errors:** For certain, typically rare, error patterns, the decoder does not detect that the error and maps the incoming n-bit data block into a k-bit block that differs from the original k-bit block.

How is it possible for the decoder to correct bit errors? In essence, error correction works by adding sufficient redundancy to the data block. The redundancy makes it possible for the receiver to deduce what the original block was, even in the face of a certain level of error rate.

Error detection and correction can be expressed as follows. The received codeword $\mathbf{c}'$ is multiplied by the transpose of $\mathbf{H}$, that is $\mathbf{c}'\mathbf{H}^{\mathbf{T}}$. If the result is a zero vector, then no error is detected. If the result is nonzero, then the resulting vector, known as the syndrome, can be used to correct errors. The exact process for correction depends on the nature of the ECC.

An example of efficient error correcting codes are Goppa codes, which can be turned into a secure coding scheme by keeping the encoding and decoding functions a secret, and only publicly revealing a disguised encoding function that allows the mapping of a plaintext message to a scrambled set of code words. Only someone in possession of the secret decoding function can recover the plaintext. This technique is computationally hard to reverse using either a conventional or quantum computer.

An (n, k) Goppa code can correct any number of bit errors $t = (n - k)/\log_2(n)$ bits. The first scheme based on this code is by McEliece [MCEL78]. The private key consists of three matrices. A specific Goppa code is chosen, and represented by an $k \times n$ matrix $\mathbf{G}$. Also chosen are an $n \times n$ permutation matrix $\mathbf{P}$ and an arbitrary $k \times k$ invertible binary matrix $\mathbf{S}$. The public key is the matrix $\mathbf{G}' = \mathbf{SGP}$, plus the value t; the private key consists of the three matrices that are multiplied together.

Suppose $\mathbf{G}'$ is the public key of entity A, and entity B wishes to encrypt a k-bit message $\mathbf{x}$ with $\mathbf{G}'$. B sends $\mathbf{x}' = \mathbf{xG}' + \mathbf{e}$, where $\mathbf{e}$ is a random n-bit error vector with exactly t ones. After the A receives $\mathbf{x}'$, to decrypt the message, A computes $\mathbf{x}'\mathbf{P}^{-1} = (\mathbf{xG}' + \mathbf{e})\mathbf{P}^{-1} = \mathbf{xSG} + \mathbf{eP}^{-1}$. By using the decoding algorithm of the code, A can remove the error term and is left with $\mathbf{xS}$. Because $\mathbf{S}$ is invertible, A can recover $\mathbf{x}$.

A number of refinements of this scheme have been developed to reduce key size [SEND18].

Multivariate-Based Cryptographic Algorithms

Multivariate schemes are based on the difficulty of solving systems of multivariate quadratic polynomials over finite fields. The term multivariate polynomial refers to a polynomial in more than one variable, and the term quadratic polynomial refers

to a polynomial of degree 2. In general, these schemes can be described as follows. The public key consists of a set of m polynomials:

$$P(x_1, \ldots, x_n) = (p_1(x_1, \ldots, x_n), p_2(x_1, \ldots, x_n), \ldots, p_m(x_1, \ldots, x_n))$$

which can be expanded to the following:

$$p_1(x_1, \cdots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{1,ij}x_i x_j + \sum_{i=1}^{n} p_{1,i}x_i + p_{1,0}$$
$$p_2(x_1, \cdots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{2,ij}x_i x_j + \sum_{i=1}^{n} p_{2,i}x_i + p_{2,0}$$
$$p_m(x_1, \cdots, x_n) = \sum_{i=1}^{n}\sum_{j=1}^{n} p_{m,ij}x_i x_j + \sum_{i=1}^{n} p_{m,i}x_i + p_{m,0}$$

with m equal to the number of equations and n equal to the number of variables.

In general terms, encryption using the public key is performed as follows: Given a plaintext $m = (y_1, \ldots, y_n)$, the ciphertext is:

$$P(m) = (p_1(y_1, \ldots, y_n), p_2(y_1, \ldots, y_n), \ldots, p_m(y_1, \ldots, y_n)) = (c_1, \ldots, c_m)$$

The private key is the inverse mapping P^{-1} and provides the plaintext:

$$(y_1, \ldots, y_n) = P^{-1}(c_1, \ldots, c_m)$$

The assumption is that given P it is difficult to find P^{-1}, but not vice versa. More specifically, the security of the scheme depends of the difficulty of the following problem. For a given $P(x_1, \ldots, x_n)$, find a vector $(z_1, \ldots, z_n)$ such that $P(z_1, \ldots, z_n) = \mathbf{0}$.

The digital signature is formed in a similar way. A hash of message m is computed that can be expressed as $\mathrm{H}(m) = (h_1, \ldots, h_n)$. Given a signature $(s_1, \ldots, s_n)$ for m, the signature can be verified by testing if $\mathrm{H}(m)$ is equal to $P(s_1, \ldots, s_n)$.

We give two simple examples for multivariate polynomials over the finite field $\mathrm{GF}(2^2)$. The only irreducible polynomial of degree 2 for this field is $x^2 + x + 1$. Table 14.8 shows addition and multiplication mod $(x^2 + x + 1)$. The polynomials in the field are symbolically represented by the integers. Suppose the public key consists of:

$$p_1(x_1, x_2, x_3) = 1 + x_3 + 2x_1x_2 + x_3^2$$
$$p_2(x_1, x_2, x_3) = 2 + x_1 + 2x_2x_3 + x_2$$
$$p_3(x_1, x_2, x_3) = 1 + x_2 + x_1x_3 + x_1^2$$

Given a 6-bit message 010000001 represented as $(x_1, x_2, x_3) = (2, 0, 1)$, encryption of the message with the public key is performed as follows:

$$p_1(2, 0, 1) = 1 + 1 + (2 \times 2 \times 0) + (1 \times 1) = 1$$
$$p_2(2, 0, 1) = 2 + 2 + (2 \times 0 \times 1) + 0 = 0$$
$$p_3(2, 0, 1) = 1 + 1 + (2 \times 1) + (2 \times 2) = 1$$

The ciphertext is $(1, 0, 1)$.

Table 14.8 Arithmetic in GF(2^2)

Polynomial	0	1	X	$x + 1$
Binary representation	00	01	10	11
Integer representation	0	1	2	3

(a) Polynomial Representation

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

(b) Addition

×	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

(c) Multiplication

Now consider the public key:

$$p_1(x_1, x_2, x_3) = 1 + x_2 + 2x_0x_2 + 3x_1^2 + 3x_1x_2 + x_2^2$$
$$p_2(x_1, x_2, x_3) = 1 + 3x_0 + 2x_1 + x_2 + x_0^2 + x_0x_1 + 3x_0x_2 + x_1^2$$
$$p_3(x_1, x_2, x_3) = 3x_2 + x_0^2 + 3x_1^2 + x_1x_2 + 3x_2^2$$

Suppose a message m with hash value $\mathrm{H}(m) = (1, 2, 3)$. The owner of the private key matching the above public key generates the signature (0, 0, 1). The signature can be verified by a recipient by generating $\mathrm{H}(m)$ and encrypting the result with the public key. In this case, the calculation yields:

$$p_1(1, 2, 3) = 1 + 3 + (2 \times 1 \times 3) + (3 \times 2 \times 2) + (3 \times 2 \times 3) + (3 \times 3) = 0$$
$$p_2(1, 2, 3) = 1 + (3 \times 1) + (2 \times 2) + 3 + (1 \times 1) + (1 \times 2) + (3 \times 1 \times 3) + (2 \times 2) = 0$$
$$p_3(1, 2, 3) = (3 \times 3) + (1 \times 1) + (3 \times 2 \times 2) + (2 \times 3) + (3 \times 3 \times 3) = 1$$

which verifies the public key.

The public-private key construction can be described in a way that is similar to that of code-based schemes. The process starts with an easily invertible quadratic map F: $K^n \rightarrow K^m$. For the public key, the structure of this mapping is hidden by combining F with two invertible maps S: $K^m \rightarrow K^m$ and T: $K^n \rightarrow K^n$. The public key P is the composed map $S \circ F \circ T$:$K^n \rightarrow K^m$. The private key consists of the three maps.

A number of variations on this basic scheme are currently being pursued [DING17].

Hash-Based Digital Signature Algorithms

To get an idea of how a hash-based signature algorithm works, we first consider a scheme proposed by Lamport [LAMP79]. Assume a hash function that produces a b-bit hash value. Thus, for SHA-256, $b = 256$. In Lamport's scheme, a public/private key pair is used only once for a given message m. The steps involved are as follows:

1. Compute the b-bit hash value $\mathrm{H}(m)$.
2. Generate $2b$ secrets bit strings, two for each bit location k in $\mathrm{H}(m)$, $S_{0,k}$ and $S_{1,k}$. The set of secret values constitutes the private key.

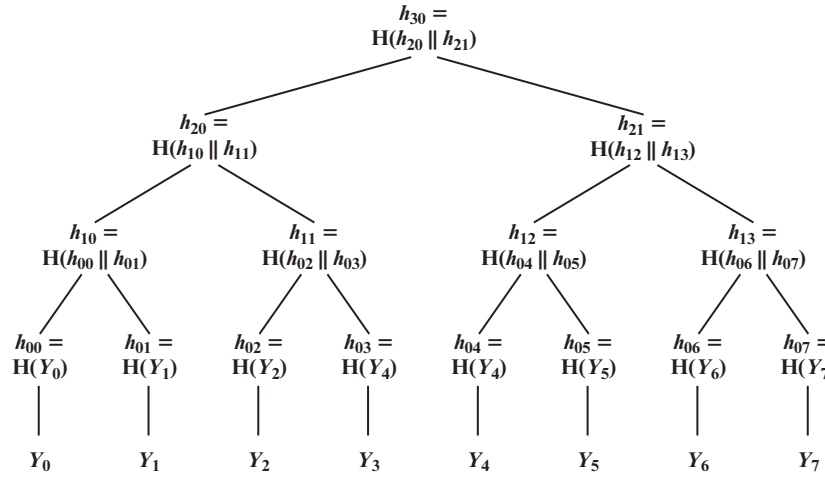
3. The public key consists of the hash values of each secret value: $H(S_{0,k})$, $H(S_{1,k}), k = 1, \ldots, b$.
4. The digital signature consists of half of the hash values calculated in step 3. For the block m, the signature is generated as follows. If the kth bit of $H(m)$ is 0, then the kth element of the signature is $S_{0,k}$; if the kth bit of $H(m)$ is 1, then the kth element of the signature is $S_{1,k}$. Thus, the signature reveals half of the private key.
5. Signature verification involves the following: The verifier calculates $H(m)$. Then the bits of $H(m)$ are used to pick out the corresponding elements of the public key. So, if the kth bit is 1, select $H(S_{1,k})$. Then the b hashes in the signature are compared to the b hashes selected from the public key. If all match, the signature is verified.

There are a number of drawbacks to this scheme. The signing of a message reveals half of the private key. This is not enough to allow an attacker to sign additional messages with different digests, but it would not be secure to use this key pair more than once. Further, both the public and private keys are of considerable length.

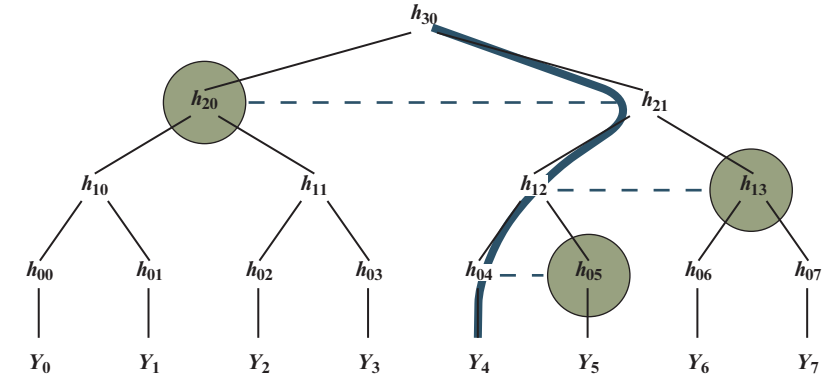
Merkle [MERK79] proposed a technique that builds on the Lamport scheme, using the concept of a hash tree. It allows a signer to precompute a number of public-private key pars that can be used to generate signatures that can all be verified with the same public key. And the long-term public key need only be the size of a hash value. For this scheme, a tree of hash values is constructed. The scheme allows the signing of a number of messages $N = 2^n$, where n is an integer. The signer generates N private keys X_i with $0 \leq i \leq 2^n - 1$, and computes the corresponding public keys Y_i. Each public key is the concatenation of the $2b$ hash values described in step 3 of the Lamport scheme. Then a hash tree is formed. Each node of the tree is labeled $h_{i,j}$, where i denotes the level of the node and corresponds to the distance of the node from a leaf. Thus, a leaf of the tree is level 0 and the root of the tree is level n. Figure 14.14a shows a tree for $n = 3$.

The tree is constructed in pairs, starting with the leaves. Each leaf consists of the hash of one of the public keys. For higher levels, each pair of values at one level is concatenated to form a double block, and the hash of that block is computed. This process continues until a single value results, known as the Merkle root. The Merkle root becomes the single public key, to be used to verify up to N signatures. This has two advantages: the public key is quite small, and it can be used for multiple signatures.

The process for signing a message m_i is as follows. First the Lamport digital signature LS_i is generated from $H(m_i)$, as before, consisting of the set of b secret strings $S_{j,k}$ for $j = 0$ or 1, $k = 1, \ldots, b$. The value LS_i forms part of overall digital signature for the message, which is available for verification. However, in this scheme, the verifier does not have possession of the Lamport public key Y_i, so this must be supplied as part of the signature. Further, the verifier must be able to authenticate that this public key Y_i is valid. To do this, the verifier needs to trace a path from the leaf to the root and confirm the root value. For this calculation, the verifier needs every node on



(a) Merkle tree with eight leaves



(b) Merkle tree with authentication path

Figure 14.14 Example of Merkle Hash Tree

the path, plus the value of the brother node at each level; these are also provided in the signature. Putting this all together, the signature S_i for m_i consists of the following:

$$S_i = (LS_i, Y_i, h_{0,x}, h_{1,x}, \ldots, h_{n-1,x})$$

where x equals either $l + 1$ or $l - 1$ at each level l of the tree.

An example should make this process clear. Figure 14.14b shows the Merkle tree with eight leaves and the path to the root from Y_4. To authenticate the public key, the verifier computes

$$H(H(H(H(Y_4) \| h_{05}) \| h_{13}) \| h_{20})$$

If this value equals the public key h_{30}, then Y_4 is authenticated. Once Y_4 is authenticated, it can be used to verify the signature Y_4.

An important drawback of Merkle-related schemes is that the signer must keep track of which onetime signature keys have already been used. This can be difficult in large-scale environments. Stateless variants are a matter of current research [BUTI17].

14.5 KEY TERMS AND REVIEW QUESTIONS

Key Terms

constrained device	lightweight cryptographic algorithm	post-quantum cryptographic algorithm
cryptoperiod	lightweight cryptography	post-quantum cryptography
cryptosystem	microcontroller	quantum computing
deeply embedded system		quantum safety
embedded system		

Review Questions

14.1 Define embedded system.

14.2 Define constrained device.

14.3 List and briefly explain three classes of constrained devices.

14.4 What are the chief design constraints for lightweight cryptographic algorithms?

14.5 What are the typical characteristics of lightweight cryptographic algorithms?

14.6 What are the main types of cryptographic algorithms for which lightweight cryptography is relevant?

14.7 Briefly explain the rationale for post-quantum cryptography.

14.8 What are the main types of cryptographic algorithms for which post-quantum cryptography is relevant?

14.9 List the four main mathematical approaches being studied for post-quantum cryptography.