Appendix 10
Practical Implementation in Telecommunications and Data Storage

Physical cryptographic methods are only possible in symmetric cryptography. They therefore relate to the security properties of confidentiality and integrity, and to a limited extent also to authenticity. Confidentiality involves data encryption, while integrity involves the MAC (Message Authentication Code).

When physical methods are discussed today, the focus is almost always solely on the generation and distribution of cryptographic keys. This approach is far too narrow, because much more is required for data encryption and MAC determination in the context of telecommunications and data storage. The generation and distribution of keys alone are just one component among several necessary ones.

Therefore, this book examines the entire complex. This is primarily because a narrow focus on the generation and distribution of cryptographic keys fails to account for important aspects that are, in fact, crucial even for the generation and distribution of the keys themselves.

1 Data Encryption and MAC in Telecommunications

In modern telecommunications protocols, certificate-based authentication occurs first, followed by key generation and distribution, and then data encryption and/or MAC calculation. The Diffie-Hellman algorithm (PKCS #3: Diffie-Hellman Key Agreement Standard) is typically used for key generation and distribution, today mostly in its classical form or based on elliptic curves (e.g., using Bernstein's Curve 25519) and in the future as a quantum-computer-secure mathematical variant, e.g., based on the Crystal Kyber (ISO/IEC 27001:2022). This is followed by data encryption, usually using the AES algorithm (ISO/IEC 18033-3). With Diffie-

Hellman, a random number is used on both sides in the first step, which must be sufficiently secure on both sides. The randomness of the key, which is derived from these two random numbers, depends on the quality of the better of the two random numbers. A key only needs to be available during an ongoing telecommunication session and can be deleted immediately after transmission.

1.1 Physical methods

In DV-QKD and CV-QKD, for cost reasons, typically only one terminal is used on each side, which means that one side always acts as the transmitter and the other as the receiver. As a result, one side primarily determines the randomness of the key and thus the security of the data encryption, which is often unacceptable in practice for security reasons, especially if the two communication partners do not belong to the same company or organization. Furthermore, two communication partners who each possess only a transmitter or a receiver terminal must never interact with one another.

However, this issue, which can compromise IT security, does not arise in QKD with entanglement, in RKD, or in MKD. In MKD, both sides must generate the random numbers to ensure equivalence between the two communication partners and transmit the random numbers to the other side.

How data encryption is performed depends on the possible key length. With QKD, current market offerings typically allow for only a single mathematical algorithm (such as AES-256), particularly for long distances and large data volumes. While the cryptographic key can be changed very quickly in QKD (e.g., a key change every minute or second), this improves security only marginally. With RKD, only a mathematical method can generally be used. Only MKD with a one-time pad consistently enables—practically independent of data volume and distance—physical methods from key management through to the cryptographic security objectives of confidentiality (via data encryption) and integrity (via MAC calculation).

For MAC calculation (ISO/IEC 9797-2), all methods where the MAC is calculated using an XOR function are particularly suitable, such as GCM (ISO/IEC 13157-3).

1.2 Extending OpenSSL

OpenSSL is a toolkit for general cryptography and secure communication[1] . Its functions can be accessed directly via the command line or integrated into an application. To this end, OpenSSL provides two libraries, "libcrypto" and "libssl," for developers to implement cryptographic functions and enable secure communication with their application.

The "libssl" library contains various secure network communication protocols, such as SSL/TLS, DTLS, and QUIC. This library depends on "libcrypto" for its cryptographic procedures[2].

The "libcrypto" library contains a wide range of cryptographic algorithms that are also used in various Internet standards. It provides APIs for cryptography, such as encryption, digital signatures, and hash functions. Furthermore, supporting APIs for cryptography-related standards and other non-cryptographic APIs are provided[3].

For the individual cryptographic algorithms used by "libcrypto," there are various implementations in so-called providers. "libcrypto" is shipped with five providers (Default, Base, FIPS, Legacy, Null). However, providers can also be created and made available by third parties[4] .

As explained in the source referenced above, libssl implements only the standardized cipher suites (see, for example, RFC-8446 Section B.4). Therefore, in projects that use libssl+libcrypto, a custom implementation can only be used if it is part of a standardized cipher suite. These are defined by the IANA.

However, it is possible to implement custom algorithms, as shown at https://github.com/provider-corner/vigenere.

1.3 Websites

To secure modern website access with new algorithms, several terms must be defined. Two technologies particularly worth mentioning are the web browser, which displays the website, and the web server, which provides the data. Communication occurs in layers; relevant to this implementation are TCP connections and QUIC (based on UDP; this layer enables, among other things, guaranteed delivery and protection against attacks—QUIC should be understood as a combination of UDP+TLS).

---

[1] ossl-guide-introduction
[2] ossl-guide-libssl-introduction
[3] ossl-guide-libcrypto-introduction
[4] ossl-guide-libraries-introduction

These connections also use TLS to protect data in transit—this is referred to as HTTPS. Websites are transmitted via HTTP. HTTP is known in versions 1.0 (one page per IP), 1.1 (virtual hosts by name), 2.0 (binary communication and pipelining), and 3.0 (QUIC as the basis and 0-RTT TLS).

HTTP/3 is gaining in importance[5] . With HTTP/3, out-of-order processing becomes a challenge for cryptographic implementation[6] —it is important to note that QUIC uses different keys for the data stream and header encryption[7] .

Proxies are another technology worth mentioning. Proxies process HTTP requests and forward them. Well-known examples include forward proxies, which forward a user's request to the Internet, and reverse proxies, where requests from the Internet are terminated and forwarded to one or more servers. Only HTTP can be transparently proxied; with HTTPS, the connection must always be terminated, and encryption must be negotiated with or through the proxy. Proxies are also used, for example, to access the Tor network.

1.4 TLS in Web Communication Components

The mantra "Don't roll your own crypto" is widely accepted. Software vendors also follow this approach by implementing cryptography using established software libraries.

Type	Tool	Library	Source
Browser	Firefox	NSS	https://qubip.eu/transition-of-nss-and-firefox-to-support-the-quantum-secure-internet-browsing/
Browser	Chrome/ Chromium	BoringSSL	https://chromium.googlesource.com/external/gob/boringssl/boringssl/
Server	Apache2	OpenSSL	https://httpd.apache.org/docs/2.4/ssl/
Server	Nginx	OpenSSL	https://nginx.org/en/docs/http/ngx_http_ssl_module.html
Server	Node.js	OpenSSL	https://nodejs.org/download/release/v9.10.1/docs/api/tls.html#tls_tls_ssl
Server	ligHTTPd	Several, including OpenSSL	https://redmine.lighttpd.net/projects/lighttpd/wiki/Docs_SSL

[5] HTTP/3 traffic is steadily increasing https://w3techs.com/technologies/details/ce-http3
[6] https://datatracker.ietf.org/doc/html/rfc9001#name-receiving-out-of-order-prot
[7] https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-34#name-header-protection

Server	Caddy	(golang) crypto/tls	https://github.com/caddyserver/caddy/blob/master/modules/caddyhttp/reverseproxy/httptransport.go
Server	Microsoft IIS	MS SChannel	https://techcommunity.microsoft.com/blog/iis-support-blog/about-https-schannel-tls-capi-ssl-certificates-and-their-keys/815200
Tool	Curl	Various, including OpenSSL	https://curl.se/docs/ssl-compared.html
Tool	wget	GnuTLS or OpenSSL	https://www.gnu.org/software/wget/manual/html_node/HTTPS-_0028SSL_002fTLS_0029-Options.html
Proxy	Squid	OpenSSL or GnuTLS	https://www.squid-cache.org/Doc/config/tls_outgoing_options/
Proxy	HAProxy	OpenSSL	https://www.haproxy.com/documentation/haproxy-runtime-api/reference/set-ssl-tls-key/
Proxy	Traefik	(Go) crypto/tls	https://github.com/traefik/traefik/blob/master/pkg/tls/cipher.go
Proxy	Envoy	BoringSSL	https://www.envoyproxy.io/docs/envoy/latest/faq/build/boringssl
Proxy	g3proxy	Various, including OpenSSL	https://github.com/bytedance/g3?tab=readme-ov-file#:~:text=TLS%20over-,OpenSSL,-/%20BoringSSL%20/%20AWS%2DLC

However, modern browsers do not allow non-standard TLS ciphers to be enabled (https://security.stackexchange.com/a/70772), while web servers sometimes do allow this[8] . Based on the documentation, it is reasonable to assume that g3proxy also supports all ciphers from OpenSSL in both the upstream and downstream via tls_client.

It should also be noted that many SSL stacks are interconnected. For example, OpenSSL was long the standard, but due to poor maintainability, the LibreSSL project and BoringSSL[9] emerged from it.

It should also be noted that OpenSSH can also function as a SOCKS proxy and that OpenSSH also uses OpenSSL[10] . With such a tunnel, selected connections (not just HTTPS) could be protected by OpenSSL, regardless of routing decisions or network monitoring.

[8] Regarding nginx: see `ssl_conf_command` or ciphers such as CAMELLIA256 (no longer widely used in browsers today, but included in implementations due to RFCs)
[9] https://www.haproxy.com/blog/state-of-ssl-stacks
[10] https://www.openssh.org/releasenotes.html

1.5 Is TLS on websites end-to-end?

However, TLS protection should not be understood exclusively as an end-to-end connection. Companies often use (transparent) proxies to monitor browsing behavior or reduce bandwidth consumption through caching. Websites use Content Delivery Networks (reverse proxies, distributed to be close to the end user, often with filtering functions, but outside the control of the end user or website operator) to distribute the load. CDNs are very popular (approx. 44% of the most frequently visited websites use CDNs[11]) and must therefore not be ignored . Consequently, especially for larger websites or across corporate networks, it can no longer be assumed that HTTPS is end-to-end.

Legend: {} mandatory component, [] optional component

{Browser} <-> [Corporate Proxy] <-> [CDN] <-> {Web Server}

1.6 How can the impact of OpenSSL be maximized?

To cover as much of the communication as possible, a web server such as nginx should be used. Because browsers do not use OpenSSL, a proxy such as g3proxy should be used on the client side. This ensures that communication across trust boundaries is secured.

2 Data encryption and MAC for on-premises data storage

Data is typically stored in databases or files. Role-based or attribute-based access control systems are usually employed for access control. These systems determine rights such as read, write, and delete permissions, etc. In conjunction with cryptography, cryptographic access control systems are required; in role-based or attribute-based systems, these systems only recognize encryption/decryption rights, from which read permissions are then directly derived (reading is only possible through successful decryption). For speed reasons, only symmetric methods such as AES or the one-time pad are suitable for encryption/decryption and MAC calculation. Unlike in telecommunications, the keys must be retained for as long as the data encrypted with them is stored and relevant for a read operation. And that can be many years.

[11] https://webtechsurvey.com/technology-type/content-delivery-network

The required keys are determined based on roles and attributes. In a role-based system, each role is assigned a key, which usually originates from the role's administrators (role managers). When authorized users leave a role, a new key must be generated by the role's administrator. Once all data relevant to that role has been re-encrypted, only the new role key is required. In the other case, which is common in decentralized systems, all older role keys must also be retained. In this case, the number of required keys increases constantly. Over the years, this can result in a very large number of keys, all of which must be stored with high security over an extended period of time.

2.1 Physical methods:

Because the keys may be needed for a very long time, it is generally necessary to store them securely and for the long term at all points where encryption/decryption takes place, which is inherent to the MKD system and therefore relatively easy to implement.

In one of the solutions for MKD described in Chapter2.2 , the keys are generated by an administrator in that role, and a copy is created for each authorized user for encryption/decryption. This copy is delivered to all authorized users using a portable storage medium suitable for MKD. The administrator alone determines the key quality. If multiple administrators jointly determine the key, an XOR function of these keys is possible, and the key quality can be distributed among multiple administrators (role managers).

QKD and RKD cannot be used for this purpose because direct distribution to n different authorized users of the role is not possible.

For data encryption, the one-time pad can always be used upon request—as can mathematical methods, of course—because only MKD can be used and sufficient key material can always be provided here.

2.2 MKD solution with LISA:

Currently, only one product is available on the market, called LISA ([12]), which fully meets the processes described in Chapter2.2 and the requirements listed above. LISA is available for PC operating systems and was developed, among other things, to enable the full use of MKD on desktop PCs, laptops, tablets, etc., for data storage and telecommunications.

[12] www.insitu.software

There is also a solution where the same conventional PC (desktop, laptop, tablet) can operate in a multisession mode with different security levels at the operating system level: ranging from open to completely restricted communication (local, restricted, or no network and Internet communication possible).

As shown in the figure below, based on an open, secure operating system, various security zones are created for individual applications with their different requirements. Depending on the requirement profile, different restrictions are implemented within the containers. For example, low security requirements allow access to the Internet. Medium security requirements allow access to internal resources but not to the Internet. Furthermore, depending on security requirements, different encryption methods can be used (ranging from AES-256 to the One-Time Pad with provably 100% security). The security zones are isolated from one another and from the underlying operating system. This minimizes security risks.

<missing figure>

Figure 6: Bit Resolution

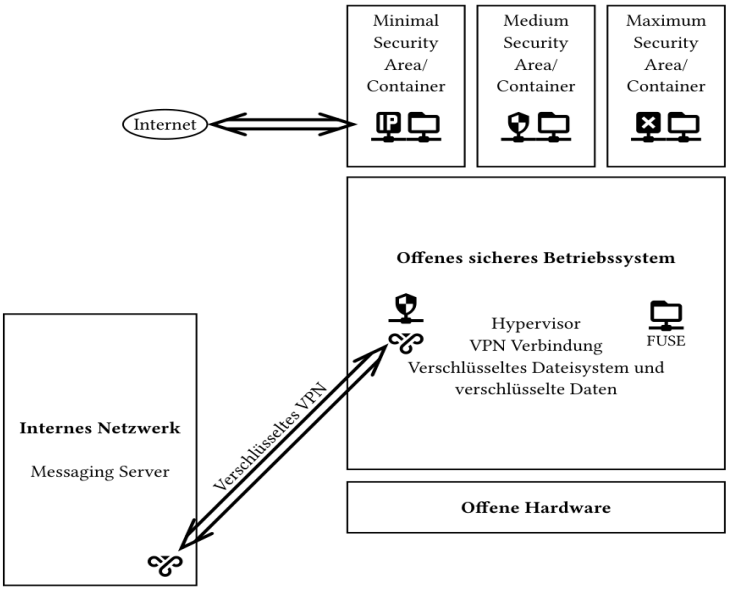

Figure 11

LISA can also be used in conjunction with a suitable HSM if the PC cannot be adequately protected, if trust is insufficient, or if the rules or policy do not permit PC use. In this case, the entire implementation of MKD and data encryption using

the one-time pad takes place in the HSM, and the PC provides the additional environment required for data storage via LISA.

A novel cryptographic access control system that operates on a role-based basis ensures that control over the data lies exclusively with the authorized parties and that authorization is granted on-site. Data storage can, in principle, continue to take place in a cloud, as this—even at the database level—is encrypted on an authorization-based (role-based) basis, and the encryption—like the entire cryptography used—can also be performed purely using physical methods. For highly sensitive data, LISA offers end-to-end math-free cryptography through the full implementation of MKD technology, using a one-time pad for encrypting and decrypting data. This applies not only to telecommunications but also, on a role-based basis, to all files and individual elements of the rows and columns in a database. And the entire process occurs automatically and reliably in the background through integration into PC operating systems—with or without an HSM.

Optionally, however, mathematical cryptography can also be used for less sensitive data by generating the keys in highly secure smart cards (according to ISO/IEC 7816) and encrypting them specifically for each TPM chip on PCs or a user smart card. On the PC, the keys are then decrypted by the TPM chip or the user smart card solely for use. AES-256 is used for encryption and decryption. In principle, AES-based main memory encryption is always performed as well. LISA therefore allows, depending on security requirements, for each role to specify whether data is left unencrypted, encrypted with AES-256 (i.e., using a mathematical algorithm), or encrypted with a one-time pad (i.e., consistently using physical methods based on MKD). These three parallel options are also important for temporary files (such as those in Microsoft Office), which are encrypted, for example, with AES-256, while the actual files are encrypted with the one-time pad.

To implement a one-time pad in combination with a non-deterministic key, new key bits are always used whenever a file or database fields are encrypted. Each role has its own key file containing the necessary key bits (e.g., 1 TB of key bits per key file). For each file and every individual data field in a database, a note is made indicating the bit address in the key file from which the encryption originated. This ensures that when the data is read—which may not occur until years later—exactly the same key bits can be retrieved and used in the one-time pad. The key file for each role is created by a role administrator by saving a non-deterministic random number of suitable length (e.g., 1 TB), generated by a suitable random number generator, to portable storage media suitable for MKD. All individuals authorized for read access receive an identical copy of this random number personally (or via a trusted third party, etc.) using portable storage media suitable for MKD. When all key bits in a key file have been used up, the file can be replenished with a sufficient number of key bits using an extension key file. The length of a key file can be freely selected by role administrators (ranging from a few gigabytes to terabytes).

3 Summary

MKD with a one-time pad enables a completely math-free solution (except for XOR) for telecommunications and data storage, i.e., only physical methods are used. QKD and RKD are only suitable for telecommunications, and since current market offerings typically combine them with a mathematical encryption method such as AES-256 (Advanced Encryption Standard). With the new encryption methods OTPH and XTSC, very large amounts of data are also possible, and therefore data storage applications as well.