University of Amsterdam

Practical Security and Key Management

Research Project 2

Author:
Magiel van der Meer

Supervisors:
Jeroen van der Ham - Uva
Marc Smeets - Linq42

Abstract

In the contemporary world where a still growing quantity of communications uses the Internet, it becomes more and more important to secure these communications. A lot of possibilities to secure an arbitrary communication channel are available but there may be even more methods to weaken the security of the channel. To secure private communications over public computer networks cryptography is applied. Despite the availability of cryptographic software libraries performing all the math, successful cryptography is non-trivial to implement and requires constant maintenance.

The goal of this paper is to aggregate public available information to define the current best practices on practical security and key management. The paper focuses on three elements to improve the available documentation on practical security and key management. The first is *Key Management* in general, the second describes *Personal Communications* and the third explains *Client/Server Communications*. For these three elements, another three levels of security are defined. These levels cover from little to no security to high secure environments with defined policies and budget available. After conducting the desk research, it can be concluded that a lot of information on the covered subject is available but scattered around the Internet and information is not always trustworthy because the information is outdated, superseded, or just plain wrong.

For individuals with the need to secure their communications, it is recommended to make sure they use a recent source and do as much as possible background research on the topic. Companies dealing with customer - privacy sensitive - data should regularly consult with an external IT security specialist to keep their system up to date against the latest tread.

July 2014

Contents

1 Introduction

Nowadays the encryption of communications between entities is more and more important. Encryption allows communications over untrusted computer networks ensuring confidentiality, integrity and authenticity of the data in transit. Confidentiality of data means only the intended sending and receiving entities can read the data, while data integrity ensures the data is not modified after being sent by the sending entity. Authenticated data is data provably coming from the expected sending entity and not an intruder in the network path.

As more and more formerly offline services move to online Internet services, more information of people which is meant to remain private becomes potentially vulnerable in larger numbers. The vast amount of users on the Internet makes the Internet a popular target for abuse. Services on the Internet are actively being scanned by malicious users looking for, among others, credit card numbers, online banking login information and health or social security information. This information is then abused, often for financial profit. Access to these personal details should be protected by entity verification, encryption and data integrity to reduce the costs and consequences for the victims.

Concepts

Confidentiality of data is achieved by encrypting it using known encryption schemes with a secret key known to the communicating entities. Common examples of encryption schemes are Data Encryption Standard (DES)[12] and the Advanced Encryption System (AES)[6].

Integrity of data is ensured by using a Message Authentication Code (MAC) based on cryptographic algorithms like Secure Hashing Algorithm (SHA) or Message Digest (MD5). A MAC is a irreversible representation of the input text, often represented as a hexadecimal number of a certain length. It is based on the fact that a given input will always render to the same cryptographic output which can be compared at the sender and receiver. Any mismatching in the resulting MACs violates data integrity.

Authenticity may refer to two concepts; entity authentication and data authentication. Entity authentication validates both entities involved that they are who they claim to be. Data authentication ensures that data sent by one entity actually originates from this entity. Entities might authenticate each other based on shared knowledge - a *pre-shared key* or *symmetric-key authentication* - or trusting the same third party, often a so called Certificate Authority (CA), who should have verified and trusts both entities. The latter is known as *public key authentication* or *asymmetric-key authentication*.

Encryption can be described as the process of converting information, the *plaintext*, to an unrelated output called the *ciphertext*. Decryption is the opposite of encryption, converting the ciphertext back to the plaintext. An encryption *cipher* is a set of algorithms used by the encryption and decryption processes. The cipher uses a *secret key*, ideally only known to the sending and receiving entities, to encrypt or decrypt a given text.

Keys are used by an encryption algorithm to perform encryption and decryption operations. The key length of an encryption algorithms key is measured in bits. The security of an algorithm

is also represented in bits and is relative to the key length. The security of an algorithm cannot be longer (stronger) than the key lengths. However, it can be shorter than the key length. This is possible due to reduction attacks on the algorithm. For instance, a key with a length of 128 bits with a known attack which only needs 96 bits to decrypt the content has an effective security of 96 bits.

Two key algorithms can be distinguished; *symmetric keys* and *asymmetric keys*. When using a symmetric key, the key has to be known to all entities who need to access the data. A symmetric key is also called a pre-shared key. The pre-shared key is used for both encryption and decryption of data. Examples of pre-shared key encryption are (3)DES and AES. Asymmetric key encryption, or public key encryption, uses a pair of two keys, one public key and one private key. Public key encryption is explained more in-depth in section 6.3.

Key exchange is a problem existing for a long time. How does one share a decryption key with an other entity without possibly compromising the key to untrusted third parties? Several solutions are available, each suitable for different scenarios. A mathematical solution is the concept of Diffie-Hellman (DH)[9] key exchange in which two entities rely on a mathematical formula which outputs the same number (private key) based on public shared information. Yet it is not possible for an attacker to calculate the same number based on the information publicly transmitted, although this solution is vulnerable to Man-in-the-Middle (MitM) attacks. An alternative for DH is Elliptic curve Diffie–Hellman (ECDH)[1].

As described in asymmetric key encryption, it is also possible to securely exchange a key using a common trusted CA. Another key exchange mechanism is the web-of-trust which avoids the use of a central CA. Entities who want to participate in the web-of-trust are responsible themselves for retrieving public keys of other entities in a secure manner.

Signatures are a cryptographic representation of the signed data. The signature is created using a private key and can be verified by an another entity by using the corresponding public key.

RSA is a public key cryptosystem used for secure data transmission. It is named after its inventors Rivest, Shamier and Adleman who also founded the RSA company in 1982. The algorithm offers encryption, decryption, signing and verification of signatures. Rivest, Shamier and Adleman (RSA) is an asymmetric key system and thus uses a public and a private keypair. The algorithm is based on the factorization of large integers; which in essence mean that multiplying p and q (both random, prime and of similar bit-length) to modulus n is easy while computing p and q from n is not possible in an realistic time frame[2]. The length of modulus n is the key length.

Digital Signature Algorithm (DSA) is a signature algorithm based on the discrete logarithm problem. DSA is a standard proposed by U.S. National Institute of Standards and Technology (NIST) in 1991 and can only sign and verify content. Applications often offer encryption with DSA but the ElGamal[3] encryption scheme is used in that case. A variant of DSA is Elliptic Curve

[1] `https://en.wikipedia.org/wiki/Elliptic_curve_Diffie-Hellman`
[2] `https://en.wikipedia.org/wiki/RSA_(cryptosystem)#Key_generation`
[3] `https://en.wikipedia.org/wiki/ElGamal_encryption`

Digital Signature Algorithm (ECDSA). This is the same principle as DSA except it uses Elliptic Curves. It is believed the bit size needed for ECDSA is about twice the size needed for the security level in bits. For example, at a security level of 80 bits the size of a DSA key is at least 1024 bits, whereas the size of an ECDSA public key would be 160 bits while the signature size would remain the same for both DSA and ECDSA[4].

Eliptic Curve Cryptography is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields. The first suggestion of using Eliptic Curve Cryptography (ECC) in cryptography was by Neal Koblitz in [18] and Victor S. Miller in [21]. ECC based algorithms entered the practical world of cryptography in 2004 and 2005. Multiple curves are defined for usage in cryptography by NIST but are considered back-doored by the National Security Agency (NSA) [14].

Client/server communications often use asymmetric key encryption with the necessary key exchange. An example is a secure web server using HyperText Transfer Protocol Secure (HTTPS) to protect the data between the client and the server. More on client/server security in section 6.

Key management is an important concept of cryptography. Exposed private keys will allow an attacker to impersonate other entities and listen in on secured traffic between entities. Working with private keys in a safe manner requires a lot of prerequisites and considerations. In section 4 a set of guidelines is given on proper key management.

Research

The technical terms described above are only a small summary of what is necessary to apply security which actually has an added value. The Internet is a rich source of information ranging from background information to hands-on guides used for quick implementation. But without a background in IT security it is hard to distinguish good information from out-dated or plain wrong information. This paper tries to aggregate information on several methods often used in security with the aim to become the source with relevant, checked, documented and up-to-date information on practical security. Subjects touched involve key management, secure personal communications and methods to secure client/server communications.

Goal of this project is to address these issues and create one place to go to for all information necessary for safe key management in various situations. The case described above leads to one, general research question:

How to combine practical security and secure key management by aggregating relevant public available information?

Points of interest around this question are:

- What elements should be secured?
- How secure should these elements be?
- How can this level of security be reached?

[4] `https://en.wikipedia.org/wiki/Elliptic_Curve_DSA`

What elements should be secured?

Before one can apply security on an element, it should be clear what this element is. This paper defines three secure elements explained in section 2.

How secure should these elements be?

For unique scenarios a different level of security might be desirable. E.g. a big company working with financial transactions will have different requirements and budget than a individual who is only interested in protecting his privacy from his Internet Service Provider (ISP) or government. This paper will define three security levels which will be cross-referenced with the secure elements. The levels are not carved in stone and should only be used as an example; it is possible to combine recommendations from several levels to suit specific needs. The security levels are further defined in section 3.

How can this level of security be reached?

A set of practical guides to apply the concepts outlined in 2 and 3 is given in section 8.

Acronyms

AES Advanced Encryption System
CA Certificate Authority
CRL Certificate Revocation List
CSR Certificate Signing Request
DANE DNS-based Authentication of Named Entities
DER Distinguished Encoding Rules
DES Data Encryption Standard
DH Diffie-Hellman
DNS Domain Name System
DNSSec Domain Name System Security Extensions
DSS Digital Signature Standard
DSA Digital Signature Algorithm
ECC Eliptic Curve Cryptography
ECDH Elliptic curve Diffie–Hellman
ECDHE Elliptic Curve Diffie–Hellman Ephemeral
ECDSA Elliptic Curve Digital Signature Algorithm
ETSI European Telecommunications Standards Institute
FQDN Fully Qualified Domain Name
HA High Availability
HMAC keyed-Hash Message Authentication Code
HSM Hardware Security Module
HKPS HTTP Keyserver Protocol over HTTPS
HSTS HTTP Strict Transport Security
HTTP HyperText Transfer Protocol
HTTPS HyperText Transfer Protocol Secure
IETF Internet Engineering TaskForce
IP Internet Protocol
IMAP Internet Message Access Protocol
IM Instant Messaging
IMAPS Internet Message Access Protocol Secure
IPsec Internet Protocol Security
ISP Internet Service Provider
MAC Message Authentication Code
MD Message Digest
MitM Man-in-the-Middle
MSD Mean Shortest Distance
MDA Mail Delivery Agent
MTA Mail Transfer Agent
NSA National Security Agency
NaCl Networking and Cryptography library
NIST U.S. National Institute of Standards and Technology
OCSP Online Certificate Status Protocol
OOB Out-Of-Band
OTR Off-The-Record
PEM Privacy-enhanced Electronic Mail
PKCS Public-Key Cryptography Standards
PFS Perfect Forward Secrecy
PGP Pretty Good Privacy
PKI Public Key Infrastucture
RPKI Resource Public Key Infrastructure
RSA Rivest, Shamier and Adleman
SHA Secure Hashing Algorithm
SMTP Simple Mail Transfer Protocol
SMTPS Simple Mail Transfer Protocol Secure
SSH Secure SHell
SSHFP Secure SHell FingerPrint
SSL Secure Sockets Layer
TLS Transport Layer Security
URL Universal Resource Locator
VPN Virtual Private Network

2 Secure elements

Security differs per unique environment. When securing communications, the environmental variables should be known. This paper distinguishes three categories, or elements, in which security can (and should) be applied. These elements should cover the bigger part of IT security returning in day to day scenarios.

2.1 Scope

The first element is "Key Management". Working with cryptographic keys requires knowledge, planning and depending on the environment a budget. The second element is "Personal communication"". Personal security involves communications like e-mail and Instant Messaging (IM). The last element described is the "client/server" model. Applications like web and mail servers might transmit sensitive information and this should be secured by applying encryption.

The defined security levels might share the same security principles and mechanisms. The global workings of each principle are explained here and references are given to more in-depth information about inner workings.

The research was limited to these three elements due to time constraints. Other implementations as Domain Name System Security Extensions (DNSSec) and Virtual Private Network (VPN) exist but are not discussed in this paper. Both topics are large enough to be subject for research themselves. New projects like Protonmail and Mailpile are given short attention to estimate their added value to this research. A short paragraph is written on them.

2.2 Key management

Key management can be defined as a set of (repeating) tasks and considerations around working with keys. Key management should not be considered a one-time-setup-and-forget task. Successful key management involves returning tasks and continuous attention. This paper provides possible solutions to work with keys in a secure way. Key management involves the points listed below. These points are discussed more in-depth in section 4.

- Generation
- Keys
- Key back-up
- Escrow
- Historic data
- Access
- Usage
- Publication
- Revocation & Rollover

2.3 Personal communications

PGP offers authenticity and integrity by signing and/or encrypting a message from a sender to one or more receivers. PGP uses public key encryption (briefly explained in section 6.3) to achieve these goals. PGP also incorporates non-repudiation: after the message transmission it is still provable that the message is sent by the person identified by the signature attached in the message. In practice, PGP is mainly used in e-mail transmissions and signing files being distributed over insecure channels like HTTP. A guide containing best practices for PGP can be found in section 8. In this paper the following concepts regarding PGP are described.

- Generation & key safe-keeping
- Key algorithm and length
- Role separation
- Expiration
- Publishing
- Rollovers
- Revocation
- Web-of-trust

S/MIME is a protocol similar to PGP but with a low adoption rate. The Internet Engineering TaskForce (IETF) responsible for the standardization of S/MIMEv3 protocol states the project has a minor adoption rate [15]. Therefore, S/MIME is not covered by this paper.

Off-The-Record (OTR) is a cryptographic protocol that uses AES, DH and SHA to provide strong encryption for instant messaging. OTR offers Perfect Forward Secrecy (PFS) and deniable authentication, as opposed to PGP. A more extended explanation of OTR is available on [20]. OTR will not be discussed in this paper because it is used "under-the-hood" by instant messaging applications and configuration depends on the implementation by the application. Example applications using OTR are Cryptocat, Pidgin and Adium.

Protonmail From: `https://protonmail.ch`

> ProtonMail was founded in summer 2013 at CERN by scientists who were drawn together by a shared vision of a more secure and private Internet. Early ProtonMail hackathons were held at the famous CERN Restaurant One. ProtonMail is developed both at CERN and MIT and is headquartered in Geneva, Switzerland. We were semifinalists in 2014 MIT 100K startup launch competition and are advised by the MIT Venture Mentoring Service.
>
> Messages are stored on ProtonMail servers in encrypted format. They are also transmitted in encrypted format between our server and users' browsers. Messages between ProtonMail users are transmitted in encrypted form completely within our secured server network. Because they never leave our secured environment, there is no possibility to intercept the encrypted messages enroute.

Protonmail claims to offer encrypted message exchange between users. Messages seem to be encrypted between the clients browser and the Protonmail systems and are likely to reside there in encrypted form. When exchanging messages between Protonmail users a message is visible in the other user's inbox. However, when sending encrypted messages to non-Protonmail users they receive a link leading to the encrypted content. The sender still needs another safe channel of communication to communicate the decryption key to the receiver. Protonmail hides any encrypt and decrypt actions from the user by offering a thickbox "Encrypt this message". The user can not use PGP to encrypt mail to external users. Since Protonmail is still under heavy development, it will not be further discussed in this paper.

Mailpile From: `https://mailpile.is`

> Mailpile is email software (an app) that runs on your desktop or laptop computer. You interact with the program using your web browser. The goal of Mailpile is to allow people to send e-mail in a more secure and private manner than before. With Mailpile, your e-mail is downloaded from the Internet (via an email server POP3 / IMAP), and stored locally on the computer where Mailpile is running. Mailpile is an

email client used for encrypting/decrypting emails and not an email service. Mailpile does currently not issue its own email accounts/addresses. You can use your existing email address on top of Mailpile. If your existing email service stores your emails on their servers, then they will not be able to read the encrypted emails. The Mailpile client is used for sending/receiving, managing, organizing and storing emails on your computer/USB/cloud.

Whereas Protonmail is a hosted solution, like Gmail, Mailpile is a mail client which needs to be run by the user. It has PGP functionality integrated in the core of the system enabling the user to send and receive encrypted content with any other system using the PGP standard. Mailpile is also still in development and lacks key-features like support for Internet Message Access Protocol (IMAP). Mailpile will also not be further discussed in the paper due to its immaturity.

2.4 Client/server security

In the client/server model multiple cryptographic protocols are available. The protocols most used are Transport Layer Security (TLS) and its predecessor Secure Sockets Layer (SSL). These two protocols differ in cryptographic properties but this paper considers them the same when no specific version of the protocol is mentioned in context. Another protocol to encrypt data between two computer nodes is Internet Protocol Security (IPsec). Only TLS and SSL are discussed in this paper since there are no significant application using IPsec other than IPsec VPN.

Transport Layer Security is used in server-side applications like web servers (Nginx, Apache2), mail related servers (Postfix, Sendmail, Dovecot) and secure remote access providers (OpenVPN). Several implementations of TLS are available; the most common used libraries are OpenSSL, PolarSSL and GnuTLS. Microsoft's Windows has its own implementation called Secure Channel (or SChannel). Applications are often build on top of one of these libraries but, depending on the application, can be recompiled using another library.

For applications using TLS/SSL a lot of configuration options are available. The default options might favor backwards compatibility over security while this is not necessarily the best situation for the operator of one of the services. A overview of best practices per service is given in section 8.

3 Security levels

Creating truly secure environments is hard - if not impossible - to achieve. This paper defines three levels of security, *basic*, *medium* and *high*. Each level increases the security but also the skill level and costs needed to implement. For distinct scenarios' implementations one of the given scenarios might be most applicable or secure enough. An example of an environment using basic security is a small web company selling products online while processing minimal personal data of customers. A bank handling big financial transactions and providing remote access to employers will probably be an example of the high security environment. The given guidelines in 'Practical' are not carved in stone. Details may vary based on requirements like the need for PFS, non-repudiation or plausible deniability. Elements of individual levels can of course be combined to create a level suitable for specific needs.

Basic

The basic security level can be suited for individuals having the need to send and receive signed and/or encrypt e-mail messages. Examples might be security enthusiasts worrying about the actions of their government or their e-mail provider reading and storing their messages. Companies dealing with less sensitive customer data can use of this level as well.

Medium

The medium security level can be identified by individuals with special interest in extensively securing their communications and the systems they use to communicate with. An example might be journalists in countries with repressive regimes or IT security researchers. Companies dealing with more sensitive customer data and even company secrets might use this level.

High

The high security level might be best described by banks. Companies dealing with privacy sensitive and financial customer data should use stronger security measures opposed to the subjects described in the Basic and Medium levels. This level includes comprehensive investments in security like using a Hardware Security Module (HSM), personal hardware tokens and predefined procedures for certificate revocation and/or rollovers.

4 Key management

4.1 Generation

To keep a private key truly private, it is important to generate a keypair on a trusted system. Several options are available, all based on the desired level of security.

Basic The least secure way to generate keys is to use an online[5] system on which the keypair will be used. This system might have been compromised with malware looking for key generating processes and weaken them or export the generated private keys without the end user noticing.

Medium A more secure method to generate keys is to use a system running a live Linux distribution (e.g. Tails-OS [4]) which has never been online before. The Tails-OS is a live environment installable on an USB thumb drive which can be used for this purpose. In effect every operating system can be used as long as it is kept offline, but Tails-OS is pre-equipped with the right tools to generate a master key without an initial connection to the Internet. This ensures a pure system which is not compromised in any way. The keys can be exported from the live environment using storage media like USB thumb drives, CD-roms or a smart card. External systems like the USB device Yubikey [31] or the smart card Gemalto IDBridge K30 [11] can be used for storing keys.

The Yubikey allows the generation and storing keys, but doesn't allow exporting of private keys [30]. This can be considered a good thing - the private key cannot be leaked - but it also cannot be backed up. Losing access to the Yubikey would result in losing access to all encrypted data and detaches the online identity one has carefully built. The K30 smart card can store a pre-generated keypair (up to 3 times 4096 bits) and allows for exporting the key.

Ideally the private key is never made available on a system which has been connected online in the past. The Tails-OS live image offers specific pre-installed tools for secure key management.

High The method described in the medium security level is time consuming and prone to (human) errors but affordable for an individual or a small organization. For the high security level, HSM devices might be used. Depending on the device, this is either an add-on PCI(e) card or a standalone network system running specific software. An HSM aims to keep the private key safe inside the system. An HSM usually offers private key operations to other systems by means of the Public-Key Cryptography Standards (PKCS)#11 defined by RSA Laboratories. HSM devices are available in different price ranges and might not only be used in high security environments but also in environments with lower security requirements and a high amount of private key operations. One of the advantages of an HSM is the ability to automate private key operations in a secure manner. Which HSM to use depends on security design decisions and should be carefully considered.

4.2 Keys

As mentioned in the introduction, encryption keys have a lengths measured in bits. The strength of the encryption algorithm increases with a longer key length. But longer keys induce a higher system load. This might be undesirable in e.g. mobile phones or dedicated devices like routers.

[5] As in 'regularly connected to the Internet'

When choosing an appropriate key length, one should consider the purpose of the key and which algorithm to use.

What is the **purpose** of the keys? Which operations will be performed mostly? Who is going to perform the majority of operations and on what kind of devices? Different **algorithms** perform in different ways when signing, verifying, encrypting or decrypting. A small overview on relative performance is given in Table 1. Indications are relative to the compared and will in practice depend on the system which performs the operations. In general, RSA should used when there are more verifying operations to be performed than signing operations. However, for historic reasons DSA is often still used because of implementations and the fact that usage of RSA has been limited by patents until April 2010. An example of a situation where verification will happen more often than signing is in the DNSSec; a zone is signed periodically but verified by every client who is requesting records. The verifying clients will benefit from small signatures and fast verification operations.

	(EC)DSA	RSA
Signing	Fast	Slow
Verifying	Slow	Fast
Encrypting	n.a.	Slow
Decrypting	n.a.	Fast

Table 1: RSA vs. (EC)DSA[7]

For the use of signatures in PGP, performance differences are negligible and other considerations should prevail.

The desired **key lengths** is a result of the consideration performance versus security. NIST recommends a key length of 256 bits (using AES) for documents classified "Top secret" while this is expected to be safe far beyond the year 2030. The overhead this large key size introduces decreases speed of operations on content. In asymmetric encryption, a key length of 2048 bit is expected to be secure enough for the foreseeable future but for content which needs to be encrypted beyond 2030 a key length of at least 3072 bit is advised. The key lengths advised by NIST are shown in Table 2. Again, these advisories are based on the assumption the algorithm won't be broken.

Other advisories exist and each uses a different approach in determining the secure life time of a given key length for a given algorithm. For instance, the updated calculations of Lenstra and Verheul state that a 112 bit symmetric key is safe until 2066. The lengths provided here are designed to resist mathematical attacks; they do not take algorithmic attacks or hardware flaws into account. One thing all the advisories have in common is the minimum key length necessary today. The least conservative in this is the NIST. Choosing an appropriate key size depends more on the use case than on the environment.

Date	(A)/Symmetric	Discrete key	Logarithm group	Elliptic curve
<2030	2048/112	224	2048	224
>2030	3072/128	256	3072	256
>>2030	7680/192	384	7680	384
>>>2030	15360/256	512	15360	512

Table 2: NIST future recommended key sizes[17]

4.3 Key back-up

Backing up a private key potentially weakens the use of the key; the key is accessible in more than one (physical) location.

Basic & medium A single end user might back-up private keys using an USB thumb drive, a smart card or print them on paper. By physically separating the backup from the original private key, one prevents a burning house from destroying the original and the backup key. External locations to consider can be relatives, friends or a notary. When encrypting the backup key, one should not forget the passphrase used for the encryption.

High When using an HSM and depending on which HSM in use, two modules can be used to work in High Availability (HA) mode. Depending on the design and architecture, this does not physically separate the private keys. If private key material can be exported from an HSM depends on the module software in use. A software implementation of an HSM is SoftHSM[6]. SoftHSM is an application for Linux and stores the private keys in Privacy-enhanced Electronic Mail (PEM) format on a defined location on the hard drive. This allows for private key backups.

4.4 Escrow

Placing a key in escrow means that a private key is given to a trusted third party. An example might be an employer who demands to be able to decrypt content encrypted by an employee or trusted parties working together on the same encrypted content. To prevent an escrowed key to be used for signing operations, different keys should be used for signing and encryption. More on this in section 5.3. To escrow a key depends on specific situations and thus cannot be categorized in the defined security levels.

4.5 Historic data

Independently of the security level, encrypted data can be of historic importance. When rolling over a keypair one might consider to decrypt old data with the original key and re-encrypt with the new key. Another possibility is to keep using the old key for decryption purposes only and encrypt new data with a new keypair. The old key should be kept available for access to the historic data.

4.6 Access

Access to key material can be classified in two ways; physical and logical access. Physical and logical access should be restricted to specific people with the right authorization.

A key encrypted with a passphrase can be stored on the previously defined mediums. The key is less protected from physical access to the machine; an intruder can force himself to obtain the key by accessing the system. If the system is online the risk of exposing the key is present and this should be considered by the user.

When using a live environment while keeping the key offline at all times, logical access to the key is only possible in combination with physical access. Physical access to the hardware used for this can be regulated by keeping hardware in a vault and encrypting the contents of a previous defined medium.

[6] `https://www.opendnssec.org/softhsm/`

Administrative access to HSM can be regulated by company policy and only allowing certain administrative users to log in and perform private key management operations. It is most likely the HSM is located in a data center and physical access is regulated by the operator of the data center. Physical access procedures of a data center should be taken into account when designing a network and/or application with an HSM in place.

4.7 Revocation & Rollover

How to revoke a key depends on the way it was spread in the first place. When using the client/server architecture, a key can be revoked by modifying the Certificate Revocation List (CRL) or changing the state in the Online Certificate Status Protocol (OCSP) server. This is further explained in section 6.4.1. A key might be compromised before its expiration date or wrongfully issued. The key then needs to be revoked so it cannot be abused. Revoking a PGP public key is described in 5.6.

Rollover of public keys or certificates is often done because the certificate was compromised, nears its expiration date, the key length is considered too short for future use or a flaw in a used algorithm is found . Rollover procedures differ with the implementation. In a client/server architecture the certificate is presented to the client upon each new session and the client verifies the received certificate. Updating to a new certificate is seamless for the end-user when executed correctly by the server operator. When using Secure SHell (SSH), a rollover needs an update of the clients "known_hosts" file. More on this in section 6.7. Rolling over a PGP public key is described in section 5.6.

4.8 Publication

Depending on the environment the keys are used, distinct methods for public key publication are available. In the client/server model it is common for the server to send the public certificate to the client for verification on each new connection. This is discussed more in-depth in section 6.4. When using a web-of-trust, like with PGP, the public key of a sending entity should already be available at the receivers side.

4.9 Usage

Private key operations should not be performed on untrusted systems. For individuals goes that it might not be smart to use a private key on a public system like in an Internet cafe or a library. The live system Tails-OS can be used to mitigate the security risk of exposing a private key. In a high security environment the usage of a private key should be restricted by the logical and physical access to the key.

It is important that a private key is only used for what it is meant to be used for to prevent key information leakage. An example of a private key role separation is using a separate master key for the signing of subkeys and signing and encrypting content with different (sub)keys.

5 Pretty Good Privacy (PGP)

PGP is used for signing (providing non-repudiation and authenticity) and encryption of digital content which is sent over computer networks. It was created in 1991 by Phil Zimmermann and later standardized by the IETF in RFC 4880. Since the standardization the protocol is called OpenPGP[7]. The most used implementation of OpenPGP is GnuPG[8].

In current releases, GnuPG supports RSA and DSA for encrypting and signing. When choosing DSA, the ElGamal is used for encryption since DSA does not support encryption. The last stable release of GnuPG (version 2.0.25) does not yet support ECC. ECC support is included in the development release of GnuPG, which is version 2.1. Using ECC for signing and/or encryption needs to be supported on both sender's and receiver's side. Using an ECC key might exclude a lot of PGP senders from encrypting content for a receiver or exclude receivers from verifying a senders certificate. ECC for use in PGP is thus not discussed in this paper [25].

5.1 Generation & key safe-keeping

Secure key generation is described in 4.1. The actual process of generating a secure public/private keypair is described in [5, 16]. The author describes how to generate a keypair with subkeys and explains how to detach the master keypair from the generated system and store them separately. However, the author does not generate the keypair on a Internet-less system. For best security, one should not have the master key available on a system which is connected to the Internet and only booted when private key operations are necessary. Improvements on the generation process are given in 8.2.

Several best-practices are available for keeping the master keypair safe, sorted from what is considered insecure to more secure.

- Keeping the master keypair on the production system or an arbitrary USB stick.
- Keeping the master keypair on an OpenPGP card or Yubikey. This is described by Boot in [3].
- Installing the Tails-OS on an USB stick and performing master key operations as necessary when booting the OS on an offline and standalone system.
- Storing the private keys in a HSM. This requires all private key operations to take place in the HSM. A guide on using an HSM is available at [8].

5.2 Key algorithm and length

Choosing an appropriate key length is described in 4.2. For PGP it is important to choose a RSA key of at least 2048 bits long. As more and more successful attacks on the Discrete Log Problem are performed, there is a chance that DSA will be mathematically broken in the near future [29, 28].

5.3 Role separation

When using PGP it is important to use separate subkeys for signing and encrypting. The GnuPG software does automatically generate a subkey used only for encryption and decryption. The master keypair is thus only used for signing content. Since the mathematical functions used for encryption and signing are the same but reversed, it is theoretically possible to trick a key holder

[7] http://www.openpgp.org/
[8] https://www.gnupg.org/

to sign an unformatted encrypted message using the same key. This will result in the original plain text. The use of two separate keys for encryption and signing avoids this problem.

More reasons to separate keys exist but their validity depends on the environment. In a regulated environment like a commercial company, the companies management might want access to content encrypted by employers. In this scenario, the company might have a copy of all employers private keys to decrypt content. But this would allow the company's management to impersonate employers which is an unwanted side effect. By using separate keys the company can decrypt encrypted content but cannot sign messages impersonating its employers.

5.4 Expiration

When generating keys an expiration date has to be given. Keys will expire at the given date and will be considered invalid from that moment on. An expiration date can be considered a safety valve and should always be set at a reasonable time. For a master keypair a period of two years is considered a reasonable time while for subkeys one year can be used. Key expiration dates can be extended, even after expiration. An example of extending a key by changing the expiration date is shown in section 8.2.1. After extending a key it should be uploaded again to a key server for others to download.

5.5 Publishing

In order to verify received content, one must have access to the public key of the sender. Unconditionally trusting the attached public key is naive and insecure because everybody can generate keys and impersonate anybody. Out-Of-Band (OOB) validation of a public key is necessary before trusting the key. Usually public keys - or its fingerprint - are pre-shared on key signing parties and can be trusted afterwards. Common practice is to share the public key its fingerprint OOB and then fetch the corresponding key from a key server.

Users of PGP use a key server to publish their (sub) keys, revocation keys and other keys they trust. It is important to regularly sync with a key server to stay up to date and prevent usage of revoked keys. When refreshing collected keys it is possible for someone to listen in on communications with a key server and even the key server can learn who one is communicating with. This might be unwanted. Several solutions exist, like using a pool of key servers with the HTTP Keyserver Protocol over HTTPS (HKPS) protocol or the Parcimonie [24] script. By using a pool of key servers no server can learn all the keys one is using and the Parcimonie script refreshes single keys at random intervals, each time using a new connection over the Tor network.

Publishing a public key is often integrated in the plug-in for the e-mail client. An example on publishing a public key to a key server using the GnuPG tools is given in section 8.2.2.

5.6 Rollovers

Key rollovers mean that a keypair is renewed every N time. Key rollovers are impractical to execute for individuals but often used by organizations with mailing lists or for the signing of software packages. An example rollover policy can be found in [22]. RFC6489 [1] describes key rollovers for a CA in the Resource Public Key Infrastructure (RPKI). This is only applicable to organizations being a CA and thus not further described in this paper. When using PGP and one decides to start using a new keypair, the newly generated keypair can be signed with the old key and published to a key server. Every key that signed the old key can be contacted to update to the new key. The old key can no be revoked and only the new key should be used from there on.

5.7 Web-of-trust

In PGP the web of trust is a decentralized web of people signing each others keys indicating to others that the signer trusts the signee. The trustworthiness of a public certificated is calculated by the Mean Shortest Distance (MSD). If certificate A is trusted by W and X, the MSD formula is $\frac{Trustingcertificates}{Totalhops} = MSD$ which results in $\frac{2}{2} = 1$. When X is trusted by two more certificates Y Z the total hops grows to 6 since Y and Z are one hop further away from A. The MSD of A now is $\frac{2}{6} = 0.333$. A more in-depth explanation is given in [10].

Ab MSD value close to 1 should not be treaded as trustworthy, one should still validate if the public key does belong to the intended recipient.

5.8 Revocation

When using PGP a different approach is necessary since there is no central point hosting a CRL or OCSP. In PGP a keypair can be revoked by publishing the revocation certificate generated based on the private key. A keypair cannot be revoked if the revocation certificate is not created beforehand and the owner has lost access to the private key (hence the expiration date). When the private key is compromised a revocation certificate should be published to a key server. Since users of PGP regularly should refresh their local keyring, a revocation certificate should reach everybody who has the revoked key in their keyring. They can sign the revocation and send this to a key server.

6 Client/server security

Briefly explained, in the client/server model one server offers a service to one or more clients. Examples are a web service serving content for a client using the HyperText Transfer Protocol (HTTP) protocol or a mail server (MTA) receiving e-mail using the Simple Mail Transfer Protocol (SMTP) protocol. The HTTP and SMTP traffic can be wrapped in a cryptographic protocol such as TLS to secure the content and to authenticate the server to the client (and optionally the client to the server). Just like the defined protocols HTTP and SMTP the two entities (server and client) need to agree upon the cryptographic operations they will perform. These operations are defined in cryptographic protocols.

6.1 Protocols

A cryptographic protocol is a protocol that performs security-related tasks and applies cryptographic methods. Cryptographic protocols are used for secure transport of data over computer networks. A cryptographic protocol involves some or all of the following concepts[9].

- Key agreement or establishment
- Peer authentication
- Symmetric encryption and authentication
- Secure data transport
- Non-repudiation methods

Transport Layer Security is a cryptographic protocol which is widely spread and often used. TLS uses asymmetric cryptography to authenticate peers and exchange a symmetric key used to encrypt data transferring between the peers. A cipher suite is chosen in the TLS session initiation. The cipher suite dictates the methods used to authenticate peers, encrypt data and ensure data integrity. Current versions of TLS and SSL are TLS1, TLS1.1, TLS1.2 and SSLv3.

6.2 Ciphers

In cryptography a cipher is an algorithm used for encryption and decryption of data. It can be described as a series of steps to encrypt a given plaintext to a ciphertext. By reversing the steps, the given ciphertext can be decrypted back to the plaintext if the correct cipher key is used. The tasks a cipher suite performs are shown in Table 3. Examples of protocols used for each task are shown in the third column.

Method	Goal	Example
Key exchange (Kx)	Define shared secret key on both sides	DH, ECDHE, ECDH
Authentication (Au)	Authenticate entities to each other	RSA, DSS
Encryption (Enc)	Provide data confidentiality	AES, DES
Integrity (Mac)	Prove data is not modified	SHA, MD5
Non-repudiation (Rep)	Prove data is sent by sending entity	Digital signatures

Table 3: Cipher protocols

As TLS and SSL support multiple combinations of ciphers, resulting in multiple cipher suits, communication between a client and server can only be possible if both support a common cipher

[9]From: `https://en.wikipedia.org/wiki/Cryptographic_protocol`

suite. In modern TLS protocols the client sends it list of supported ciphers and the server picks a common supported cipher suite. It is recommended for server operators to regularly evaluate the ciphers supported by the server. For instance, the encryption cipher RC4[10] is considered broken [13]. Supporting weak or broken ciphers might not be wanted by the operator of a server. Directives on enabling or disabling certain cipher suits are given in section 8.3.1.

A string of good cipher suites to support is:

```
ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH+3DES:!ADH:!AECDH:!MD5
```

This string disables ciphers with anonymous key exchange (!ADH, !AECDH) and the weak hashing algorithm MD5. It favors ellipticity curve key exchange (ECDH) and AES encryption over the older and weaker DH and 3DES. This string of cipher suites results, depending on other variables like HTTP Strict Transport Security (HSTS), in an "A+" score on "Qualys SSL Labs SSL test"[11] while offering backwards compatibility up to Microsoft Internet Explorer 7 on Windows Vista. To support Internet Explorer 6 on Windows XP add at least "TLS-RSA-3DES-EDECBC-SHA" to the string of cipher suites.

6.3 Public key cryptography

Public key cryptography is a set of cryptographic algorithms using two keys, one public and one secret. Both keys are unique but mathematically connected. Algorithms used are based on the mathematical problems integer factorization (RSA), discrete logarithms (DSA) or elliptic curve relationships (ECDSA, ECC). The strength of public key cryptography relies on the fact that it is infeasible to compute the private key from the public key.

Signing of content usually is done with the senders private key. Others can verify the signature created by using the corresponding public key. Signing of messages or content provides for non-repudiation.

Encryption of content is done with the receivers public key. The receiver can decrypt the received content with its private key.

It is computationally easy for any device to generate a public/private keypair. Everybody can generate a public key for any domain name or entity. How does one know a received public key actually identifies the sending entity? In the client/server model the public key of a service is sent by the server to the client. To prove the public key belongs to the sending server it is attached to a *public key certificate*. This certificate contains information on the identity of the server and is often signed by a third party like a CA. The CA has verified the identity of the public key owner and has the public key certificate signed with its own private key. Because the CA's public key is available in the clients certificate storage, it can verify the signature and trust the received public key certificate.

6.4 Certificates

Certificate Authorities are organizations who have their so called "root" certificate included in the major operating systems and Internet browsers. A CA can sign other certificates, indicating they are to be trusted. When a CA signs a certificate, a chain of trust is created which can be

[10] https://en.wikipedia.org/wiki/RC4
[11] https://www.ssllabs.com/ssltest/analyze.html

presented by a server to a client. This chain of certificates can be verified by the client, thus determining if the server it is connecting to authentic.

CAs often offer the possibility to let them generate a private key. This theoretically means a CA can also give the private key to any other entity like a government. One might prefer to generate a private key and derive a Certificate Signing Request (CSR) from the generated private key as explained in section 4. The CSR can be used to request a signed public key from any CA supporting Certificate Signing Requests. The CA will probably use an intermediate certificate to sign the request. Intermediate certificates are signed by the root certificate and reduce the risk the root certificate should be revoked. The intermediate certificate is provided when a signed public certificate is requested.

Self signed are effectively the same as above, except for the fact that the client cannot verify a self signed certificate if the signing certificate is not present in its key store. It is up to the system administrator to spread a signing certificate over systems intended to verify the certificate. One should refrain from running a public service with a self-signed certificate as clients cannot successfully verify the chain of trust. An exception to this is a public service with a limit amount of end-users who can install the certificate. Services this applies to are Simple Mail Transfer Protocol Secure (SMTPS) and Internet Message Access Protocol Secure (IMAPS). In practice, this does not work for HTTPS. Web browsers tend to be more aggressive on killing secure connections than mail clients when the certificate does not match.

DNS-based Authentication of Named Entities (DANE) is a method of spreading certificate fingerprints using the DNSSec tree. A SHA256 or SHA512 MAC is included as a DNS record in the DNSSec tree and can be verified by a client receiving a certificate. This would bypass the traditional chain of trust dominated by CAs but practical implementations are scarce at the time of writing.

6.4.1 Formats

Public key certificates come in the X.509 format. X.509 is a ITU-T standard for Public Key Infrastucture. The X.509 standard defines the structure of a certificate. It binds an distinguished name, as an e-mail address or Domain Name System (DNS) entry, to a public key forming a certificate. The latest X.509 version (version 3) is defined in RFC3280 [12].

X.509 certificates are often represented as Base64[13] encoded certificates with the extension .PEM. The decoded certificate is often in Distinguished Encoding Rules (DER)[14] format with an extension .cer, .crt or .der. The PKCS#12 standard allows for a public and private key (which is passphrase protected) to reside in the same file with an .p12 extension. The formats defined by standards and what extensions are often used in practice are shown in Table 4.

Revocation Two solutions to revoke a signed public key are CRL and OCSP. With a CRL the clients verifying a certificate chain will periodically download the CRL from the URL specified in the public key's signing key. Several problems exist with a CRL.

[12] http://www.ietf.org/rfc/rfc3280.txt
[13] https://en.wikipedia.org/wiki/Base64
[14] https://en.wikipedia.org/wiki/X.690#DER_encoding

Formats defined		
Format	Extension	Example
Base64	.pem	Single key, public or private
DER	.crt, .cer, .der	Single key, public or private
PKCS#12	.p12	Public and private key
Formats encountered in practice		
Private key	Extension omitted, .key	Private key, often in Base64 format
Public key	.pub	Public key, often in Base64 format
CA public key	.cert, .crt, .pem, .ca	CA (intermediate) public key

Table 4: Certificate formats

1. The CRL is checked periodically for performance reasons and thus a revoked certificate will not propagate fast enough.
2. A CRL introduces a single point of failure; an unreachable CRL leaves revoked certificates considered valid by the clients.
3. A CRL contains all the certificates signed by that authority and it can cost quite some performance on the client to process a CRL.

In OCSP, a client can request the state of a certificate by sending an "OCSP request" to the signing authority. The signing authority replies with a signed, successful "OCSP response" when the certificate is valid. A client now is ensured the received certificate is not revoked. When the intermediate server is requesting the OCSP response from the CA on behalf of the servers client it is called certificate stapling. In the original OCSP protocol the client checks the validity of a public key at the signing CA. When using stapling with a (web)service, the server performs this check and forwards this to the client. The response from the CA is signed by the CA and thus the server cannot modify the response without the client failing to verify the OCSP result. A server can cache the OCSP response for a configurable time. This reduces the load on the CAs OCSP servers.

6.5 Public Key Infrastucture

A simple representation of a Public Key Infrastucture (PKI) is shown in Figure 1. A Certificate Authority is an organization issuing certificates to entities. If one chooses to trust a CA, one can verify all the entities signed by the CA. Root certificates of CAs are included by browsers and operating systems so users can verify received certificates against the root certificate. Everybody can build a PKI and become a CA but it is hard to get the root certificate included in browsers and operating systems. CA who want their root certificate certified and included by browsers are required to get audited by an ETSI and/or Webtrust certified auditor. European Telecommunications Standards Institute is the European organization responsible for standardization of IT within Europe. Webtrust is the North American counterpart. Both are responsible for certifying the CAs within their region.

CAs included by browsers are known as "well-known" certificate authorities. Some examples are Verisign, GlobalSign, Entrust, DigiCert and Comodo.

6.6 Web & Mail servers

Web and mail servers use TLS so the client can verify the server's identity. Yet in both situations the clients responds quite differently to an invalid server. When initiating an HTTPS connection

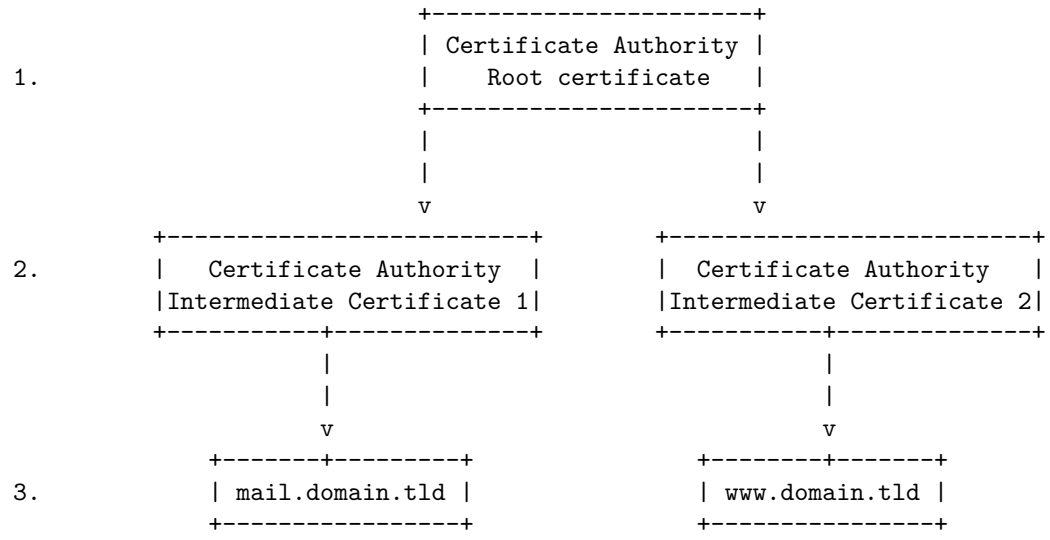

Figure 1: PKI infrastructure

to a web server with an untrusted and/or invalid certificate the browser kills the connection and depending on the implementation of the client can or cannot manually continue to the untrusted web site. In SMTP a certificate can be requested but independent of the result of the validation check, an encrypted SMTP session is initiated. This prevents an eavesdropper from listening in on the connection but with a little bit more effort the eavesdropper can launch a MitM attack and still see data being transmitted or even change contents along the way.

In both HTTPS and SMTPS it is quite uncommon for the server to validate the client. In HTTPS this would not always be of added value because of the public nature of a web server, but in SMTPS it certainly would. When a sending SMTP server is obligated to present a valid certificate it would significantly reduce the amount of spam being received since it would be impossible to send mail from an (non-)existing domain name without owning it.

Configuration examples of the HTTP server Nginx and the SMTP server Postfix are given in section 8.3.1 and 8.3.2. The examples are limited to the configuration necessary to set up and maintain a TLS enabled service.

Since a good HTTPS connection is impenetrable it is more feasible to attack the TLS initiation with a down-grade or MitM attack. Often browsers initiate the session with a web server using plain HTTP and upgrade to HTTPS later. The server can return an HSTS header telling the browser to use only use HTTPS in the future for all Universal Resource Locator (URL)s presented in the website. It then also tells the browser not to accept self-signed certificates. The next time the browser connects to the web server it will skip the HTTP part and start using HTTPS immediately. This greatly reduces the abilities a MitM attacker has.

6.7 Remote access

SSH is a method of accessing remote servers securely. It uses its own implementation of public key cryptography and thus doesn't use TLS. The most wide spread implementation of SSH is OpenSSH. Quite recent TinySSH was released. This is an implementation based on the crypto library Networking and Cryptography library (NaCl) [2] which uses only Elliptic Curve cryptog-

raphy. NaCl is designed and developed by Daniel Bernstein et al. The current state of the project is in the experimental phase.

SSH generates a fingerprint which is used for authentication of the server. A client stores the received fingerprint in a file called "known_hosts" with the server's connection name, which is either an Internet Protocol (IP) address or a DNS name. When the client detects another fingerprint, it aborts the connection and notifies the user of a possible connection hijacking attempt. This security mechanism prevents a seamless rollover of keys but greatly enhances security because fingerprint can be verified by the

Installation and configuration examples of OpenSSH including Secure SHell FingerPrint (SSHFP) are given in section 8.3.3.

7 Conclusion

A lot of information on applying security is available in the Internet. This information is often out-dated and does not always work. This paper defines the current (June 2014) best approach on key management, personal security and client/server security.

Security levels

Three security levels are defined to give guidelines on practical security. These levels are basic, medium and high. Guidelines from each level might be mixed with each other, depending on ones situation.

Key management

Security comes at a price of increased operations and knowledge. To keep security secure, end-users need some knowledge of public key cryptography. End-users generally do not have the urge to keep executing time-consuming procedures that hampers them in their work. In high secure environments, they should be educated on the consequences of failing to adhere to the security standards while keeping the standards as easy understandable as possible.

Personal security

Multiple options for secure personal communications are available. With upcoming projects like Mailpile and Protonmail it is clear there is an obvious need for communication solutions with privacy in mind. While PGP has been available for years it is still a hassle to implement and work with for non-technical minded people. It still remains a challenge to educate users of PGP in signing, publishing and revoking keys. The concepts behind PGP are not easy to grasp on and the level of adoption is low.

Client/server

While the lack of adoption of PGP is more an end-user issue, secure client/server communications have other challenges. System administrators focus on a working service and security is often neglected. Documentation for server side applications is available but not always easy to read. A lot of blog posts on configuring a web server with SSL can be found online but those often advise outdated configurations and cryptographic parameters.

Hardware Security Modules offer cryptographic operations and try to do that in a secure manner. HSM are often closed source, proprietary devices added to systems (as USB or PCI(e) device) or networks. For operating an HSM even more knowledge is a requisite and not all server side applications may work with the device. The HSM introduces a new single point of failure in the network or system.

Guidelines on keys

The following list sums up a list of minimal requirements one should adhere to when working with keys. Table 5 gives an overview of guidelines on key generation, distribution, revocation and storage plotted against the defined security levels.

- Prefer ECC algorithms over DSA and RSA
- Use RSA when ECC is not available
- Note that some believe NIST ECC curves have a back-door

- Use Ed25519 if available
- Use ECC with a key length of at least 256 bit
- When using RSA, use 2048 bit key lengths for up to the year 2030
- Use longer key lengths for communications extending after 2030
- Prefer and use AES if available
- Use AES with a key length of at least 128 bit
- Use at least SHA2-256 for digital signatures and hash-only applications
- Use at least SHA2-224 for keyed-Hash Message Authentication Code (HMAC), key derivation functions and random number generation processes
- Always backup private keys

Future work

- Keep observing progression on DSA and RSA cracking
- Think of what to do when RSA and ECC breaks
- Implement DANE as certificate validator in PGP
- Develop community-driven platform with up-to-date information on security and host this publicly available

Service:	Level	Key generation	Distribution	Revocation	Storage
PKI	**Basic**	A PKI might not be suitable here	N.A.	N.A.	N.A.
	Medium	Offline environment/HSM for generation of keys, SubCA for signing	Publish SubCA to clients	Include CRL/OCSP in SubCA	Dedicated machine for PKI operations, private keys backup at trusted 3rd party
	High	Offline environment/HSM for generation of keys, SubCA for signing	Publish SubCA to clients, Consideration: ETSI/Webtrust certification	Include CRL/OCSP in SubCA	Dedicated machine for PKI operations, private keys backup at trusted 3rd party
PGP	**Basic**	Secure generate keypair, RSA>=2048bit, sign public key	Publish public key to keyserver	Save revocation certificate separate from private key; USB drive/printed copy	On workstation
	Medium	Use offline system for keypair generation, sign public key, detach private key	Publish public key to keyserver	Save revocation certificate separate from private key; USB drive/printed copy, Escrow revocation certificate	Private key on smartcard/offline system, consider private key escrow for backup purpose
	High	Secure generate keypair on offline system, RSA>=2048bit, sign public key, detach private key	Publish public key to keyserver, consider CA-like set-up	Save revocation certificate separate from private key; USB drive/printed copy, Escrow revocation certificate	Use HSM for storage, consider private key escrow for backup purpose
HTTPS	**Basic**	Secure generate key and get CSR signed by a well-known CA, i.e. StartSSL	Distribution done by CA	Use CA available methods (CRL/OCSP)	Store key available to service
	Medium	Secure generate key and get CSR signed by a well-known CA, i.e. Gandhi	Distribution done by CA	Use CA available methods (CRL/OCSP)	Store key available to service
	High	Use HSM for key generation, get signed by a well-known CA, i.e. Verisign	Distribution done by CA	Use CA available methods (CRL/OCSP)	Use HSM for storage & private key operations
HTTPS	**Basic**	Set-up PKI, sign client CSR/generate keys	To OOB authenticated clients over HTTPS, other OOB methods	Keep copy client public key for revocation	Store keys available to service
(with client	**Medium**	Set-up PKI, sign client CSR/generate keys	To OOB authenticated clients over HTTPS, other OOB methods	Keep copy client public key for revocation	Store keys available to service
verification)	**High**	Set-up PKI using HSM, sign client CSR/generate keys	To OOB authenticated clients over HTTPS, other OOB methods	Keep copy client public key for revocation	Use HSM for storage & private key operations
SMTPS	**Basic**	Self-signed certificate	Distribution not mandatory, might use DANE for OOB validation	None	Store key available to service
	Medium	Secure generate key and get CSR signed by a well-known CA	Distribution done by CA, additionally use DANE	Use CA available methods (CRL/OCSP)	Store key available to service
	High	Use HSM or secure system for key generation, get signed by a well-known CA	Distribution done by CA	Use CA available methods (CRL/OCSP)	Store key available to service, use HSM if supported by Mail Transfer Agent (MTA)
IMAPS	**Basic**	Self-signed certificate	Distribute to managed clients	None	Store key available to service
	Medium	Secure generate key and get CSR signed by a well-known CA	Distribution done by CA	Use CA available methods (CRL/OCSP)	Store key available to service
	High	Use HSM or secure system for key generation, get signed by a well-known CA	Distribution done by CA	Use CA available methods (CRL/OCSP)	Store key available to service, use HSM if supported by Mail Delivery Agent (MDA)

Table 5: The overview

8 Practical

8.1 Key management

Generation

This section explains how to generate a private key using OpenSSL and how to get it signed by a CA.

1. 4.1: Make sure system is free of malware and/or trojans
2. 4.2: Generate a private key and save it in the current directory.

RSA Use the following *openssl* command to generate a 2048 bit RSA private key using AES128:

```
$ touch private.key
$ chmod 400 private.key
$ openssl genrsa -aes128 -out private.key 2048
Enter pass phrase for private.key: [Enter pass phrase]
Verifying - Enter pass phrase for private.key: [Enter pass phrase]
```

ECC Use the following *openssl* command to generate an ECC private key. Not all CAs might support ECC yet and not all curves are considered safe. See 1.

```
$ touch private.key
$ chmod 400 private.key
$ openssl ecparam -name prime192v3 -genkey -out private.key
```

3. Create CSR from private key and answer the questions. Make sure the Common Name matches the Fully Qualified Domain Name (FQDN) of the server.

```
$ openssl req -out request.csr -key private.key -new
Enter pass phrase for private.key: [Enter pass phrase]
[Answer questions]
```

4. 6.4: The *request.csr* can be safely sent to any CA of choice. The CA will sign the CSR with their public key and provide a signed public key along with their signing public key and optionally a intermediate certificate.
5. 6.4.1: After receiving the signed public key, one should end up with the following files for usage in an arbitrary application. The extensions and file formats might differ; it depends on the CA what they choose.

```
$ ls
private.key
request.csr
public.crt
public-ca.crt
intermediate-classx.crt
```

6. Some clients will require the *intermediate-classx.crt* file to be presented by the server to verify the chain of trust. Multiple public key certificates can be concatenated in the same file. The order of which does matter. The intermediate certificate can be omitted if not given by the CA.

```
$ cat public.crt intermediate-classx.crt public-ca.crt >> bundle.crt
```

A client verifies from the first line downwards, thus verifying *public.crt* with *intermediate-classx.crt* and the intermediate certificate with *public-ca.crt*. The *public-ca.crt* should be available in the clients public key cache.

8.2 PGP

The following commands need to be executed in order to generate a safe keypair on an offline system (System A). The last command imports the exported keypair on the production system (System B). When A is the online production system, the last command on B can be omitted.

```
A:~$ gpg --gen-key
A:~$ gpg --edit-key <thegiven@email.address>
gpg> setpref AES256 SHA512 ZLIB BZIP2 ZIP
gpg> addkey
gpg> save
A:~$ gpg --output thegiven@email.address.gpg.rev \
  --gen-revoke thegiven@email.address
A:~$ gpg --export-secret-subkeys \
  thegiven@email.address > subkeys
B:~$ gpg --import subkeys
```

8.2.1 Expiration

```
A:~$ gpg --edit-key <thegiven@email.address>
gpg> expire
gpg> expire
Changing expiration time for the primary key.
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 1y
Key expires one year from now
Is this correct? (y/N) y
gpg> save
```

8.2.2 Publishing

Publishing a public key by using the GnuPG tools on Linux

```
A:~: gpg --keyserver pgp.surfnet.nl --send-key thegiven@email.address
gpg: success sending to 'pgp.surfnet.nl' (status=200)
```

8.3 Client/server

8.3.1 Web server

Nginx is a popular web server with a lot of configuration directives with even more possible options. In the configuration box below a sample configuration is given for achieving an $A+$ in the "Qualys SSL Labs SSL test". Key directives for Nginx to actually serve documents are omitted.

To speed up PFS operations within Nginx, a DH precomputed seed file is necessary. The following *openssl* command generates this seed file. Nginx version 1.4.4 and lower relies on OpenSSL for DH input parameters. OpenSSL its DH implementation uses a 1024 bit key for the key exchange. When using a low key length for the key exchange, the 2048 bit public key will offer only a false sense of security. Thus it is important that the key used for the key exchange is of at least the same length as the public key.

```
$ openssl dhparam -rand - 2048 > dhfile.pem
```

SSL stapling is explained in section 6.4.1. Section 6.6 contains an explanation on HTTP Strict Transport Security.

The meaning of the directives is explained in the Nginx SSL documentation at [23].

```
server {
  listen                  443 spdy;
  server_name             www.example.com;
  ssl                     on;
  ssl_certificate         bundle.crt;
  ssl_certificate_key     private.key;
  ssl_protocols           SSLv3 TLSv1 TLSv1.1 TLSv1.2;
  ssl_dhparam             dhfile.pem;
  ssl_prefer_server_ciphers on;
  ssl_ciphers     ECDH+AESGCM:ECDH+AES256:ECDH+AES128:DH+3DES:!ADH:!AECDH:!MD5;
  ssl_ecdh_curve          prime192v3;
  ssl_session_cache       shared:SSL:20m;
  ssl_session_timeout     10m;
  ssl_stapling            on;
  ssl_stapling_verify     on;
  resolver                $DNS-IP-1 $DNS-IP-2 valid=300s;
  resolver_timeout        5s;
  add_header      Strict-Transport-Security max-age=63072000;
}
```

8.3.2 Mail server

Postfix is an MTA and is used to send and receive e-mail. The original SMTP specification did not contain any security; e-mail was sent unencrypted and sending and receiving entities were not verified. Later, SMTP was extended with TLS. To remain backwards compatible with SMTP hosts not supporting TLS, an receiving entity will disclose to the sending entity if it can support TLS or not after the original SMTP session is initiated. This procedure is called StartTLS. The sending entity can now issue a STARTTLS command, authenticate the remote entity and start encrypting the content to be sent.

The relevant configuration for Postfix can be split into three parts; the configuration for receiving e-mail and for sending e-mail and the part that overlaps both. The most important configuration directives are discussed here, more information can be found in the Postfix documentation [26].

Global contains configuration for ciphers to use when doing StartTLS over SMTP. This part disables SSLv2 support and forces the use of the 'high' encryption ciphers which are specified by *tls_high_cipherlist*. It is strongly encouraged to not change this setting. The **_exclude_ciphers* disables support for empty cipher suites and suites using Message Digest (MD).

```
smtpd_tls_mandatory_protocols          = !SSLv2
smtpd_tls_mandatory_ciphers            = high
smtpd_tls_mandatory_exclude_ciphers    = aNULL, MD5
smtpd_tls_auth_only                    = yes
smtpd_tls_received_header              = yes
```

Receiving configuration consists of the PKI infrastructure; the public and private keys are configured here. The **security_level* directive announces the STARTTLS parameter to sending clients and thus enables StartTLS over SMTP if the client supports this. The **ask_ccert* directive tells the server to request the client for a certificate to verify the sending entity.

```
smtpd_tls_cert_file            = public.crt
smtpd_tls_key_file             = private.key
tls_preempt_cipherlist         = yes
smtpd_tls_dh1024_param_file    = dh_1024.pem
smtpd_tls_dh512_param_file     = dh_512.pem
smtpd_tls_eecdh_grade          = strong
smtpd_tls_security_level       = may
smtpd_tls_ask_ccert            = yes
```

Sending e-mail over encrypted connections is opportunistically tried with the configuration below. *dane* 6.4 is used to verify the identity of the receiving entity. If this fails, the effective configuration is *may* and Postfix only tries to encrypt the content of the session.

```
smtp_tls_security_level     = dane
smtp_dns_support_level      = dnssec
smtp_host_lookup            = dns
```

The above combination tries to get the best of both worlds: maximum encryption and remote entity verification but maximum backwards compatibility. Other than with web servers, forcing encryption on SMTP servers will prevent one from sending and receiving mail with a huge part of the Internet [27].

8.3.3 SSH

The first time one connects to a new Unix or BSD based system the servers fingerprint is added to a local cache file. The fingerprint is a checksum of the public key file used. Every future connection will compare the cached fingerprint with the one presented by the server. This offers security to prevent MitM attacks but can be better. When publishing the fingerprint in the DNSSec tree an additional OOB verification can be performed by the client. For this to work on the server and client modifications to the default SSH configuration are necessary. This guideline assumes public key authentication is already set up correctly. A good tutorial can be found at `https://help.ubuntu.com/community/SSH/OpenSSH/Keys`. One might want to replace RSA with ECDSA and a key length of 256 bit. This guideline only uses ECDSA and disables support for RSA and DSA. Note that ECDSA support is only available in OpenSSH server and client since version 5.7. In OpenSSH versions 6.6 and higher one can choose for the Ed25519 curve by Daniel Bernstein. At the time of writing of this document, there is no standard available for publishing SSHFP records in DNSSec.

Server The SSH server generates the public/private keypair the first time it starts and saves it to "/etc/ssh/ssh_host_$algorithm_key.pub". The fingerprint of the public key is published to each connecting client so it can be used for verification.

When using the Bind-style DNS zone configuration an example record looks as follows:

```
Name      TTL    Type    Data
server    $TTL   SSHFP   3 2 e20f870e7ceab989f055d9a64b15b8a67a1bc7 [...]
```

When connecting to "server.domain.tld", the first column should contain "server" in the "domain.tld" zone. The $TTL value may be omitted, the zone's default is then used. "IN" is mandatory in Bind-notation. The "3" tells the resolving client the ECDSA protocol is to be used and the "2" indicates the fingerprint's checksum is in SHA256 format. Other possible values are shown below.

Value	Algorithm name
0	Reserved
1	RSA
2	DSS
3	ECDSA

Value	Hashing algorithm
0	Reserved
1	SHA1
2	SHA256

The following command prints the SHA256 checksum of the generated ECDSA public key.

```
awk '{print $2}' /etc/ssh/ssh_host_ecdsa_key.pub | \
  openssl base64 -d -A | \
  openssl sha256 | \
  awk '{print $2}'
```

The configuration of the domain zone should look like this where $HASH is the output of the command above:

```
$ cat zone-domain.tld | grep SSHFP
;SSHFP Records
server SSHFP 3 2 $HASH
```

Now restart the SSH server. To disable RSA and DSA support, remove the RSA and DSA lines starting with "HostKey" from "/etc/ssh/sshd_config". After disabling RSA and DSA the SSH server should be restarted again.

```
$ cat /etc/ssh/sshd_config  | grep HostKey
# HostKeys for protocol version 2
#HostKey /etc/ssh/ssh_host_rsa_key
#HostKey /etc/ssh/ssh_host_dsa_key
HostKey /etc/ssh/ssh_host_ecdsa_key
#HostKey /etc/ssh/ssh_host_ed25519_key
```

Client When published in DNS the client should be able to resolve a SSHFP record on the FQDN of the server

```
$ dig +short sshfp server.domain.tld
3 2 E20F870E7CEAB989F055D9A64B15B8A67A183A1ECFBBC7ECBB34B3F0 712BC47B
```

To tell the client to look for SSHFP records in DNSSec add the “-o VerifyHostKeyDNS=yes” to the “ssh” command on connecting or add the following lines to “ /.ssh/config” and replace the appropriate information:

```
$ cat ~/.ssh/config
Host server
  HostName server.domain.tld
  Port 22
  User user
  IdentityFile /home/user/.ssh/user-server.domain.tld
  VerifyHostKeyDNS yes
```

References

[1] G. Huston G. Michaelson S. Kent APNIC BBN, Feb, 2012. http://www.rfc-base.org/txt/rfc-6489.txt.

[2] Daniel J. Bernstein, Dec, 2013. http://nacl.cr.yp.to/.

[3] Chris Boot, June, 2014. http://www.bootc.net/archives/2013/06/07/generating-a-new-gnupg-key/.

[4] Boum, June, 2014. https://tails.boum.org/.

[5] Alex Cabal, June, 2014. https://alexcabal.com/creating-the-perfect-gpg-keypair/.

[6] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES — the Advanced Encryption Standard*. Springer-Verlag, 2002. ISBN 3-540-42580-2.

[7] Wei Dai, Mar, 2009. http://www.cryptopp.com/benchmarks.html.

[8] Sara Dickinson and Roland van Rijswijk, 2012. https://wiki.opendnssec.org/display/DOCREF/HSM+Buyers%27+Guide.

[9] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[10] Patrick Feisthammel, Oct, 2004. http://www.rubin.ch/pgp/weboftrust.en.html.

[11] Gemalto, June, 2014. http://www.gemalto.com/products/usb_shell_token_v2/.

[12] James Orlin Grabbe. The des algorithm illustrated. *Laissez Faire City Times*, 2(28).

[13] Matthew Green, Mar, 2013. http://blog.cryptographyengineering.com/2013/03/attack-of-week-rc4-is-kind-of-broken-in.html.

[14] ThomasC. Hales. The nsa back door to nist. *Notices of the AMS*, 61(2).

[15] IETF, June, 2014. https://trac.tools.ietf.org/misc/outcomes/wiki/IetfSecurity.

[16] Simon Kainz, June, 2014. https://wiki.debian.org/Subkeys.

[17] Keylength, June, 2014. http://www.keylength.com/en.

[18] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987. ISSN 0025-5718.

[19] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14:255–293, 1999.

[20] Hugo Krawczyk Mario Di Raimondo, Rosario Gennaro. Secure off-the-record messaging. 2005. http://www.dmi.unict.it/diraimondo/web/wp-content/uploads/papers/otr.pdf.

[21] VictorS. Miller. Use of elliptic curves in cryptography. In HughC. Williams, editor, *Advances in Cryptology — CRYPTO '85 Proceedings*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer Berlin Heidelberg, 1986. ISBN 978-3-540-16463-0. doi: 10.1007/3-540-39799-X_31. URL http://dx.doi.org/10.1007/3-540-39799-X_31.

[22] RIPE NCC, Jan 23, 2013. https://www.ripe.net/lir-services/resource-management/contact/ripe-ncc-pgp-key-policy.

[23] Nginx, July, 2014. http://nginx.org/en/docs/http/ngx_http_ssl_module.html.

[24] Etienne Perot, March 28, 2014. https://github.com/EtiennePerot/parcimonie.sh.

[25] Thomas Pornin, Apr, 2013. https://security.stackexchange.com/questions/34567/ecc-in-openpgp.

[26] Postfix, July, 2014. http://www.postfix.org/postconf.5.html.

[27] Postfix, June, 2014. http://nakedsecurity.sophos.com/2014/06/05/google-says-half-of-email-is-sent-unencrypted/.

[28] T. PTacek T. Ritter J. Samuel A. Stamos, 2013. https://isecpartners.com/media/105564/ritter_samuel_stamos_bh_2013_cryptopocalypse.pdf.

[29] Wikipedia, July, 2014. https://en.wikipedia.org/wiki/Discrete_logarithm_records#Finite_fields.

[30] Yubikey, July, 2014. http://www.yubico.com/faq/backup-yubikey/.

[31] Yubikey, June, 2014. http://www.yubico.com/products/yubikey-hardware/yubikey/.