Background to Modern Cryptography

Cryptography can help to keep your private correspondence private – from state surveillance, commercial harvesting of personal details, fraudsters seeking potential victims or just plain nosy parkers. It does this by *encrypting* readable *plaintext,* that you wish to keep secret, into *ciphertext* gibberish that only the recipients you choose can *decrypt* and understand.

It is not new – during Julius Cæsar's military campaign in ancient Gaul, he used a *simple cipher* to keep his plans secret. But it has grown in importance since Cæsar's time, due to modern means of communication and increasing security threats. Mathematical methods and powerful computers have come to the aid of both code-makers and code-breakers. Once the preserve of governments and the military, it is now available to businesses and the public.

This paper sets out the basic principles of modern Public Key Cryptography, with a brief account of how it got here and suggestions of where it may be going. The companion paper *Pros and Cons of Cryptography* summarises the benefits, drawbacks and costs; *Implementing Cryptography* deals with practical details of choice of software and management of cryptographic keys; and there is a *Glossary.*

Components of Cryptography

A typical cryptographic system includes:

1. The cryptography software, normally installed on your computing device as part of your software distribution, that provides standard algorithms for *general methods* such as encryption and decryption;

2. messages (both plaintext and ciphertext) that are the inputs and outputs of the cryptographic system; and

3. *keys* that are generated at your request and owned by you, and control encryption and decryption transformation of messages by the general method algorithms. According to Kerchoffs' Principle[*] "The system" (in which he does not include the keys) "must not be required to be secret, and it must be able to fall into the hands of an enemy without inconvenience." In other words, keys must be kept secret – because your security depends on them – but all other parts of the system, such as the general methods, may be open-sourced and can safely be made public.

* Article *La Cryptographie Militaire* in *Journal des Sciences Militaires* (1883)

Traditional Cryptography

Historically, cryptographic systems would decrypt a message by using the **same** key that had been used for encrypting it, which is known as a *symmetric key*. Up until the 1970s, symmetric key cryptography was the only kind available. It is fast and secure, and may be used on its own or as a component of larger systems.

Present-day symmetric key cryptography uses modern digital methods such as *AES*. They provide algorithms that can:

- **encrypt plaintext *X* by using key *K***; and
- **decrypt ciphertext *Z* by using key *K***,

where the same key, *K,* is used for both functions.

Symmetric key cryptography could provide a lightweight cryptosystem to a small group of users. The keys used by group members would be organised so that:

1. Any member can send an encrypted message to any other member;
2. therefore the sender has to know the keys of the people to whom they want to send messages; and
3. each member has their own unique receiving key. A member who receives a message that is not meant for them ought not to decrypt it.

Communication within the group would be secure against outsiders, but not against insiders unless all members take care of their keys and observe the rules. Another shortcoming is that the scheme does not provide digital signatures or signature verification. The risks can be avoided by using public key cryptography.

Public Key Cryptography

The need for privacy of communication, and later for authentication of messages and of correspondents, expanded beyond the traditional fields of commercial, intelligence and diplomatic communications. It was boosted by the two World Wars and later the Cold War against the Soviet Union, that demanded speedy decoding of enemy signals traffic and security of allied communications. Mathematicians, scientists and engineers made good progress in a series of incremental developments that culminated in the Public Key Cryptosystem.

Polish mathematicians had cracked the workings of the German electro-mechanical Enigma machine in the 1930s. From 1940 onwards, Turing, Welchman and others at the Government Code & Cipher School (now the Government Communications Headquarters, GCHQ) took up the work. Early code-breaking machines such as the Bombe were superseded by the Colossus, a true electronic computer.

In 1945 Shannon wrote a classified memorandum on the mathematical theory of cryptography, from a communication theory standpoint, providing a firm mathematical basis for later work such as that of Diffie and Hellman and of Rivest, Shamir and Adleman. A declassified version was published in 1949[¶].

In 1970 Ellis, Cocks and Williamson at GCHQ proposed a public key cryptosystem[*] that was a forerunner of both RSA and Diffie-Hellman Key Exchange (see below), but received insufficient support for development of a usable system. The work was made public in 1997.

Diffie-Hellman: a Public Key Distribution System

In the 1970s Diffie, Hellman and Merkle did a huge amount of pioneering work, inspired by their vision of a Public Key Cryptosystem. In 1976 they published the technique known as Diffie-Hellman Key Exchange (also known as Diffie-Hellman Key Agreement), based on their method of discrete logarithms[†]. This enables two parties, with no prior knowledge of one another, to generate the same secret key.

¶ See Shannon *Communication theory of secrecy systems* (1949)

* See Wikipedia article *James Henry Ellis,* report *The Possibility of Secure Non-secret Digital Encryption* (Jan. 1970) and oration *Dr Clifford Cocks CB.* (Feb. 2008).

† See Diffie and Hellman *New Directions in Cryptography* (Nov. 1976). There is also a good article on *Diffie-Hellman key exchange* in Wikipedia. The calculation uses whole number (integer) values in **modular arithmetic,** where numbers are constrained to be within a range from zero up to a limit called the **modulus.** If the result of a computation would fall outside this range, it is brought back within range by subtracting the modulus from it (or, if negative, adding the modulus to it) a sufficient number of times.

A value called the **base** that, like the modulus, has been agreed between the parties beforehand, is raised to a power, each party choosing their own value for the power and keeping it secret. The user-selected value is combined with other data so it cannot be unscrambled by an interloper or by the other party.

The parties swap numbers, so that each now holds a number, the base raised to the power of the other party's secret parameter, but from which they cannot retrieve that secret. They also still know their own secret. They raise the number they received from the other party to their own secret power, the result is the secret key

Both parties, by doing the mirror image of one another's calculation, achieve the identical result. This is because raising the base to the power a then raising the result to the power b gives the original base, raised to the power ab; the other party, by doing the exponentiations in the reverse order, raises the original base to power ba; but the two are exactly the same. No secret information had to be transmitted over a communications link.

It is not a full Public Key Cryptosystem, but it solves the key distribution problem. It is used as a component, for example in symmetric key cryptography and in the Signal, ElGamal and Elliptic-curve protocols. Merkle[‡] tried a different approach to a Public Key Cryptosystem but it was not viable on then-current hardware.

RSA: a Working Public Key Cryptosystem

In 1977 Rivest, Shamir and Adleman developed a practical implementation of a public key cryptosystem known as RSA[§]. It is now widely used, as a system in its own right or as a component of other systems such as SPAM detection in e-mail.

An essential element of a Public Key Cryptosystem is the 'trapdoor one-way function'. One-way means that it is easy to send an encrypted message, but infeasible for an eavesdropper to spy on it or tamper with it unless they possess special trapdoor information – in this case, the intended recipient's decryption key. The difficulty of devising such a function[#] was the rock on which previous attempts to implement a workable Public Key Cryptosystem had foundered. RSA provided the solution.

RSA also solved the long-standing problem of authentication. An electronic business or financial transaction needs to be backed by evidence of the grantor's consent, just like the signature on a written agreement. Now the author of an electronic document can make a digital signature, unique to themself, that the world can see is theirs but that an impostor (even the recipient) cannot forge.

Creating and verifying such digital signatures depends on the capability of decrypting plaintext. This is explained further in criterion (4) of the Criteria for a Public Key Cryptosystem section.

A digital signature authenticates the document's content as well as its author's identity, showing up any deviation of the content delivered from that which the sender sent. Signature of the document can be almost instantaneous, avoiding banking delays and circulation of papers by post or in person.

‡ See Merkle *Secure Communications Over Insecure Channels* (Apr. 1978)

§ See Rivest, Shamir and Adleman *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems* (1977).

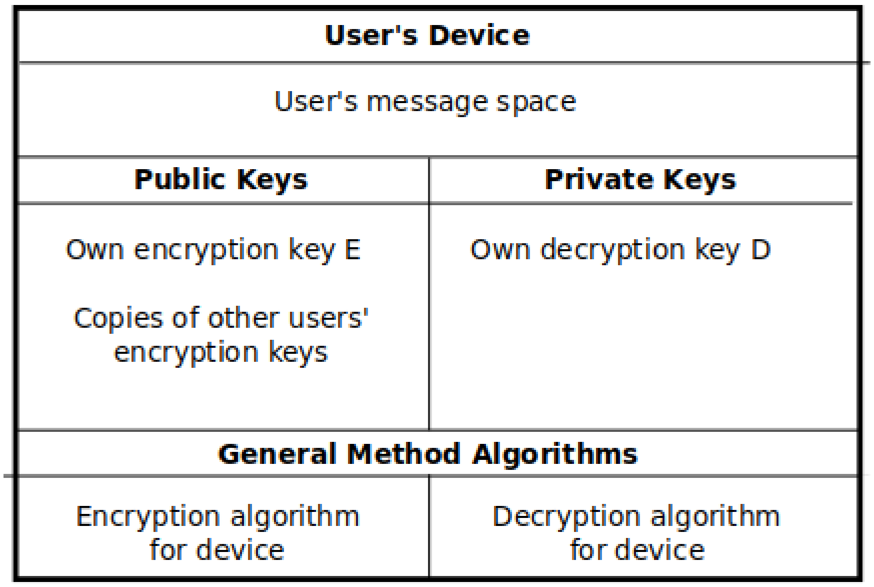
# In the RSA protocol, the encryption and decryption keys are each the product of two or more large prime numbers. Multiplying the numbers together is easy for a computer, but for an attacker to decompose the product into its constituent factors is known to be a very hard problem, making this a *one-way function.* See Rivest, Shamir & Adleman (*op. cit.* section IX.A.Factoring *n*).

A digital signature guards against a document signatory later attempting to repudiate their signature and say "I never signed that!" The proof is in the signature, signed with their key that no-one else has access to.

A Public Key Cryptosystem also simplifies key management. The user themself creates and controls their own keys and has an incentive to do it conscientiously.

Anatomy of the Public Key Cryptosystem

The diagram below illustrates a single user's Public Key Cryptosystem:


User's Device
User's message space
Public Keys
Private Keys
Own encryption key E
Copies of other users' encryption keys
Own decryption key D
General Method Algorithms
Encryption algorithm for device
Decryption algorithm for device

Each user's device is connected to the network, via which they can communicate with one another. Each contains:

– installed software algorithms for encryption and decryption;

– a message space of plaintext and ciphertext messages that may be subject to, or the result of, transformations performed by the algorithms – see criterion (2) below;

– a set of encryption keys and a set of decryption keys in one-to-one correspondence, such that each key in either set is paired with its partner in the other through their functions being the exact inverse; and

– unpaired copies of other users' encryption keys, made publicly available by their owners.

Criteria for a Public Key Cryptosystem

To qualify as a Public Key Cryptosystem, the system must satisfy the criteria¶:

1. for every key in the keyspace, there is an encryption procedure† and its inverse, the corresponding decryption procedure;

2. for every key in the keyspace and message in the message space, the encryption and decryption procedures are easy to compute;

3. for almost every key in the keyspace, it is infeasible to derive any easily computed algorithm, equivalent to the decryption procedure, from the corresponding encryption procedure. A user's encryption key can be made public without compromising the security of their secret decryption key and is therefore called a *public key;*

4. for every key in the keyspace, it is feasible to compute inverse pairs of encryption and decryption procedures from the key. If a ciphertext or plaintext message is first decrypted and then re-encrypted‡, the original message is the result. This guarantees that there is a feasible way of computing corresponding pairs of inverse transformations when no constraint is placed on what either the enciphering or deciphering transformation is to be.

> 'A function satisfying (1) – (3) of the above criteria is a "trap-door one-way function;" if it also satisfies (4) it is a "trap-door one-way permutation." … These functions are called "one-way" because they are easy to compute in one direction but (apparently) very difficult to compute in the other direction. They are called "trap-door" functions since the inverse functions are in fact easy to compute once certain private "trap-door" information is known. A trap-door one-way function which also satisfies (4) must be a permutation: every message is the ciphertext for some other message and every ciphertext is itself a permissible message. (The mapping is "one-to-one" and "onto"). Property (4) is needed only to implement "signatures".' (See Rivest, Shamir and Adleman *op. cit.*)

¶ Adapted from Diffie and Hellman *op. cit.* and Rivest, Shamir and Adleman *op. cit.*

† In this section the combined action of a key and a general method algorithm, working together, is referred to as a *procedure.*

‡ It may seem counter-intuitive to decrypt plaintext, but…
(a) We must use the **sender's** key to decrypt the signature – it is the only thing that is uniquely tied to the sender's identity; and
(b) We **can** do so because the message space is closed; we can transform the bit-string constituting the message to a different bit-string but that will still be a valid entry in the message space. (See introduction to Signing and Verifying Messages)

Operation of the Public Key Cryptosystem

Key Generation

In a Public Key system, each user is solely responsible for generating their own keys; this avoids the divided responsibility from which symmetric key systems may suffer. You should create at least one encryption/decryption pair*. You can create more than one if you wish.

We want to generate keys such that:

- it is easy for any user to send an encrypted message to any other user;
- it is easy for the intended recipient of an encrypted message to decrypt it;
- an attacker cannot derive a decryption key that would enable them to spy on the recipient's incoming mail from the recipient's encryption key; and
- an attacker cannot discover another user's decryption key by trying out each of the huge number of possible keys until they find one that works.

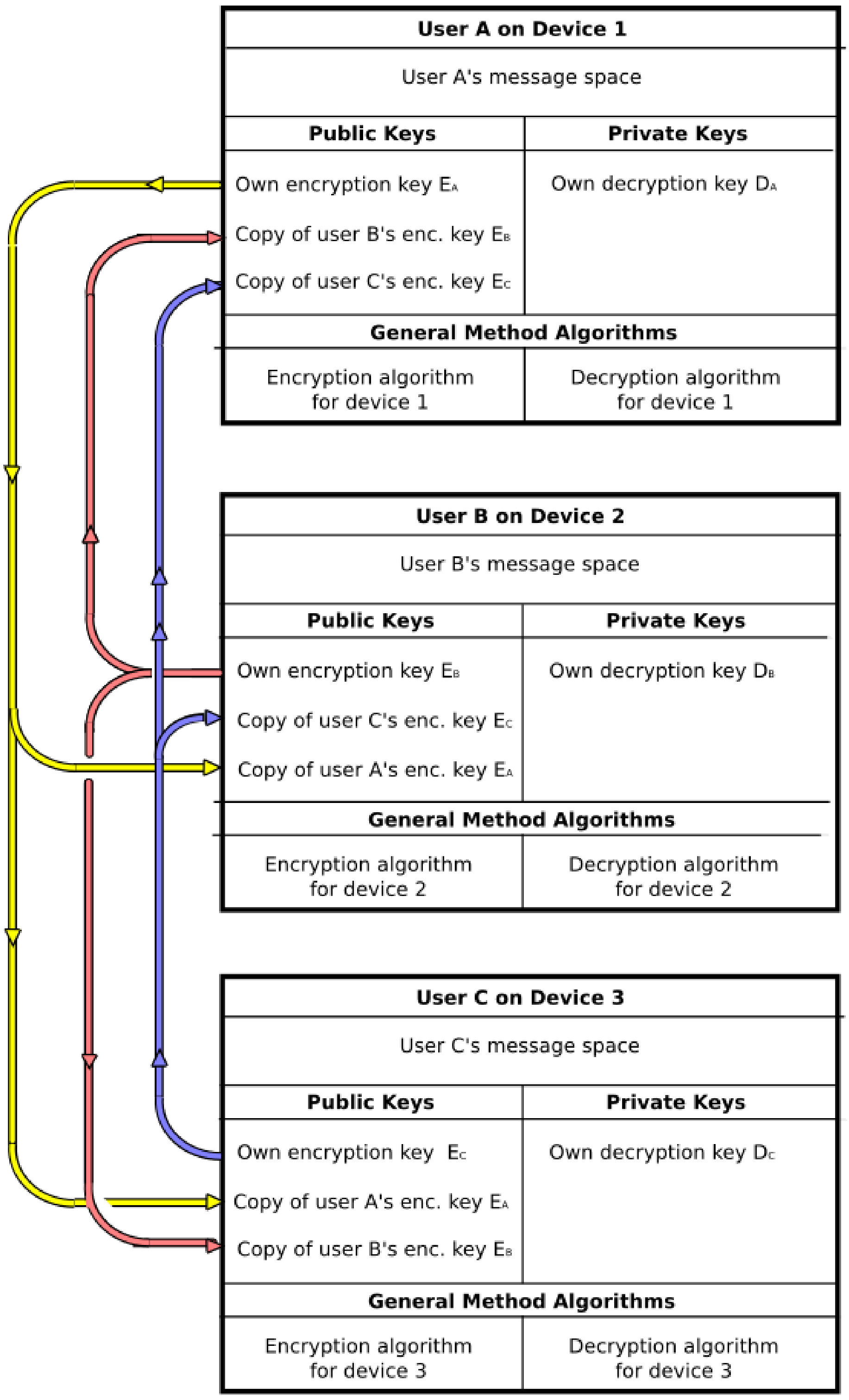
Keys are large numbers – usually a few hundred digits each. The keys are generated by software called a 'pseudo-random bit generator', which produces a stream of numbers that look random but are, in fact, deterministic; so a cunning attacker might be able to detect a pattern in the sequence and so short-circuit the trial and error process. To prevent this, it is a good idea to inject some external randomness into the process. Suggestions include timings of a noisy diode, radioactive decay or keyboard and mouse events.

Key Distribution

Users should:

- distribute a copy of their encryption key (see criterion (3) above), as shown in the following diagram, to each correspondent from whom they wish to **receive** encrypted messages;
- in return, receive a copy of the encryption key of each correspondent to whom they intend to **send** encrypted messages; and
- import those copies of other users' keys into their own key store.

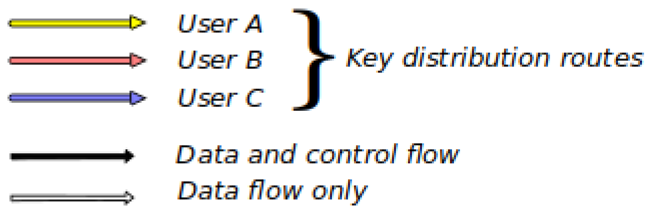
* For example, by using a utility program such as Kleopatra or Gpa


User A on Device 1
User A's message space
Public Keys
Private Keys
Own encryption key E_A
Copy of user B's enc. key E_B
Copy of user C's enc. key E_C
Own decryption key D_A
General Method Algorithms
Encryption algorithm for device 1
Decryption algorithm for device 1
User B on Device 2
User B's message space
Public Keys
Private Keys
Own encryption key E_B
Copy of user C's enc. key E_C
Copy of user A's enc. key E_A
Own decryption key D_B
General Method Algorithms
Encryption algorithm for device 2
Decryption algorithm for device 2
User C on Device 3
User C's message space
Public Keys
Private Keys
Own encryption key E_C
Copy of user A's enc. key E_A
Copy of user B's enc. key E_B
Own decryption key D_C
General Method Algorithms
Encryption algorithm for device 3
Decryption algorithm for device 3

Cryptosystems and Key Distribution

(see next page for key to symbols)

Symbols for key distribution routes, data and control flows:



Symbols apply to diagram on next page
and to flowcharts on later pages.

(intentionally blank)

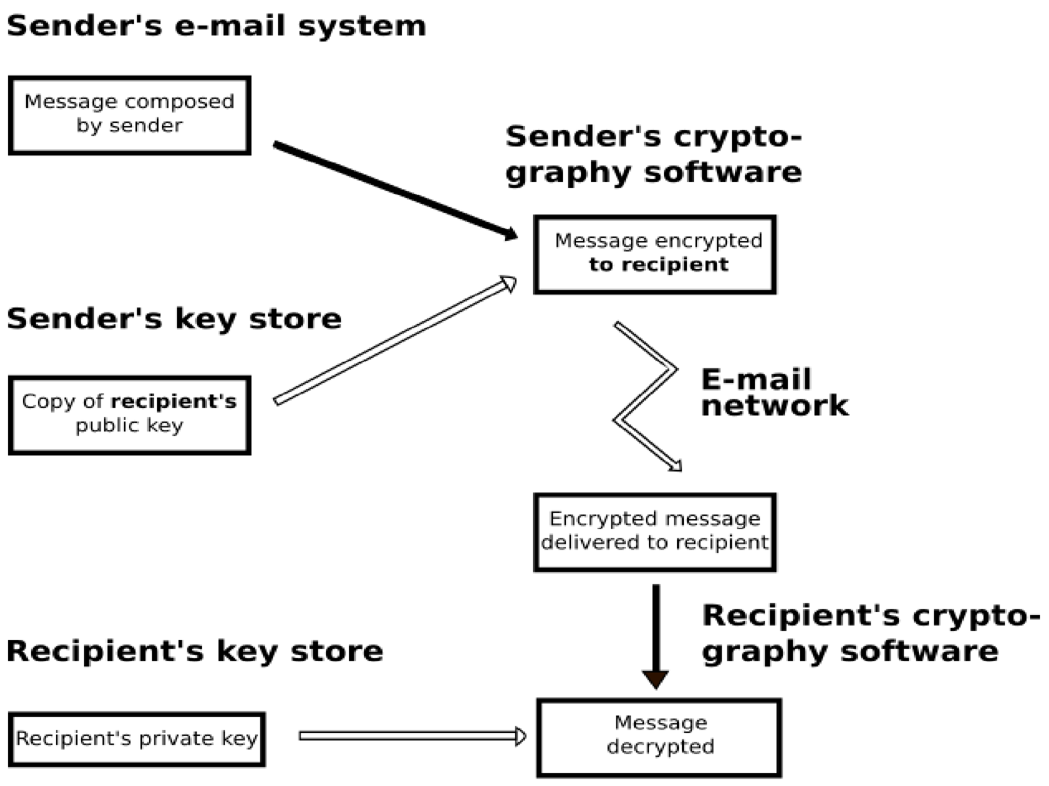
Sending an Encrypted Message

Asymmetric key methods provide algorithms that can:

– **encrypt plaintext *X* by using key *K*** ;[*] and

– **decrypt ciphertext *Z* by using key *K´***.

When Alice sends an encrypted message to Bob[†], she encrypts it using the copy of Bob's public key, ***K,*** that Bob distributed to her, and sends the resulting ciphertext to Bob. Anyone can send a message this way. Bob decrypts the ciphertext with his own decryption key, ***K '***; because of criterion (3) above, no-one but Bob – not even Alice – has access to Bob's decryption key; it is called his *private key*.


Sender's e-mail system
Message composed by sender
Sender's crypto-graphy software
Message encrypted to recipient
Sender's key store
Copy of recipient's public key
E-mail network
Encrypted message delivered to recipient
Recipient's crypto-graphy software
Recipient's key store
Recipient's private key
Message decrypted

Sending an Encrypted Message

* As an optimisation, a fast one-off symmetric key is used instead of the slower public key algorithm. It is encrypted using the recipient's public key and sent with the message.

† Alice and Bob are, by convention, names given to the actors in these examples.

Signing and Verifying Messages

As well as encrypting an electronic document to ensure its privacy, the sender can add a *digital signature* to authenticate it – just as a handwritten signature authenticates a hard copy document. For this purpose, public and private keys are used differently from the encryption case. In a communication from Alice to Bob:

– the digital signature is going to serve as proof that Alice is the true author of the document – so she alone must be able to create the signature and does so using her own **private key**[§]**,** to which no-one else has access;

– the signature must be verifiable by any recipient. Bob can use the copy of Alice's **public key** – the inverse of her private key – to re-encrypt and check the signature, as can anyone else to whom Alice has distributed her public key; and

– Alice's decryption of the signature before sending the document, and Bob's re-encryption of it after receipt, protect the privacy of the document during transit. Usually, Alice will have provided additional protection by encrypting the whole message as described in Sending an Encrypted Message above.

A fake message will not pass this test. The attacker would need Alice's private key, which they do not have, to create a convincing false signature.

Rivest, Shamir & Adleman (*op. cit.*) envisaged that the sender would decrypt the whole of the message and use that as the signature. In this case there would be no need to send the original text as well, as the recipient could obtain it simply by re-encrypting the signature. An alternative method is to compute a *hash* of the document, decrypt just the hash and send that along with the document, which must now be sent in full as the recipient cannot derive it from the hash. After decrypting the document as received, the recipient computes a hash of its content, re-encrypts the hash of the sent content given in the signature, and checks that the two hashes match.

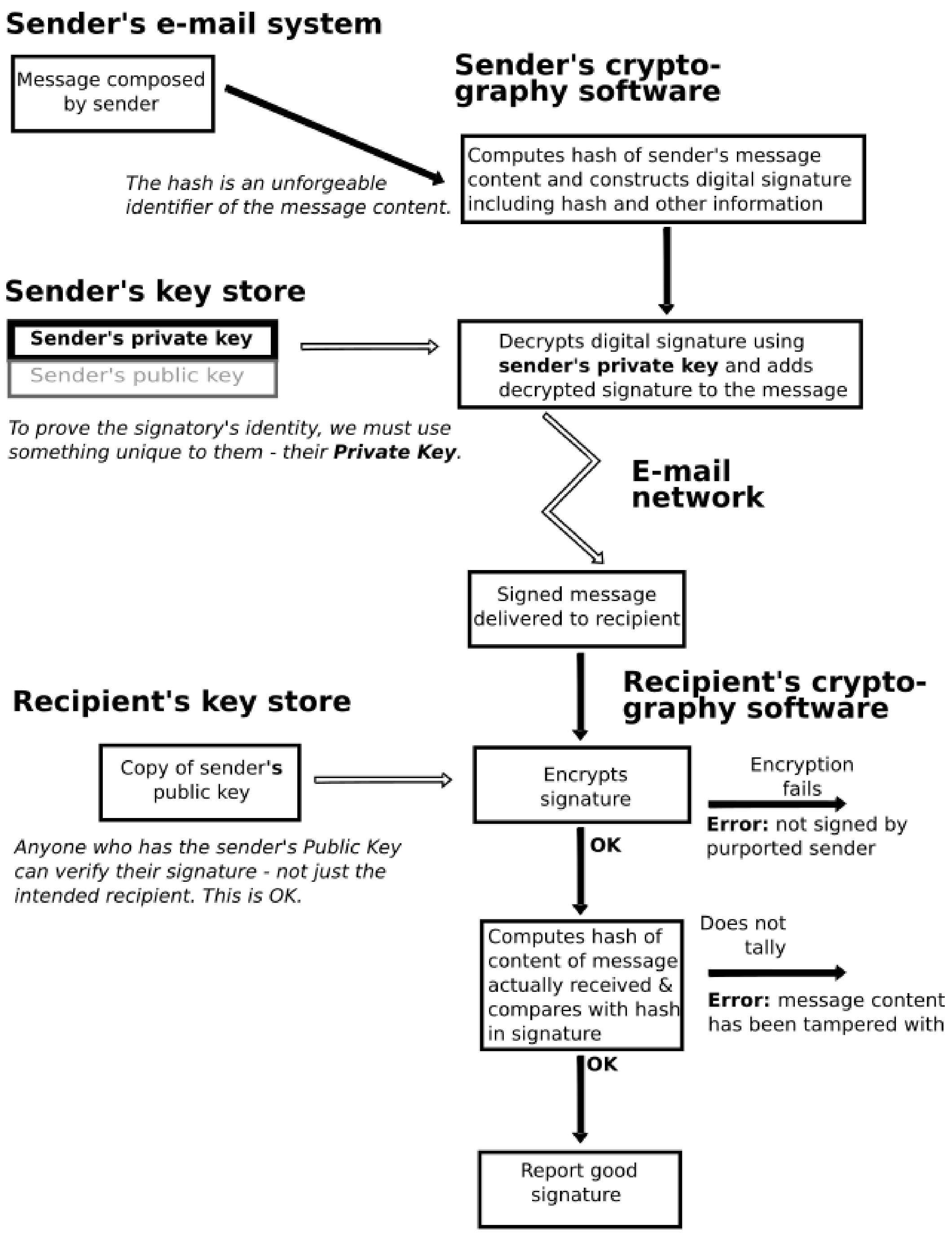
To support digital signatures and verification, the general methods must provide the algorithms:

– **decrypt plaintext signature *X* using key *K* ;** and

– **verify signature *S* by re-encrypting it with key *K´*,**

§ taking advantage of criterion (4) above, which allows her to "decrypt" the plaintext signature with her private key.

where *K* is the sender's private decryption key and *K´* is the sender's public encryption key.


Sender's e-mail system
Message composed by sender
Sender's crypto-graphy software
The hash is an unforgeable identifier of the message content.
Computes hash of sender's message content and constructs digital signature including hash and other information
Sender's key store
Sender's private key
Sender's public key
Decrypts digital signature using sender's private key and adds decrypted signature to the message
To prove the signatory's identity, we must use something unique to them - their Private Key.
E-mail network
Signed message delivered to recipient
Recipient's key store
Recipient's crypto-graphy software
Copy of sender's public key
Encrypts signature
Encryption fails
Error: not signed by purported sender
OK
Anyone who has the sender's Public Key can verify their signature - not just the intended recipient. This is OK.
Computes hash of content of message actually received & compares with hash in signature
Does not tally
Error: message content has been tampered with
OK
Report good signature

Signing and Verifying a Message

The author of a confidential message signs it digitally in their own name, to identify themself as the originator. This runs the risk of *Surreptitious Forwarding*. If the recipient of the message sends it on to someone else, no cryptographic record is made **in the message** to show that this forwarding has taken place. If the onward recipient is a person not entitled to see the message, this could be a serious breach of security for which the original author may be blamed – as theirs is the only signature that appears in the message.

A number of ways round this problem[†] have been suggested.

Adoption of Public Key Cryptography

Public Key Cryptography is used for secure web-sites (where you see a padlock icon in your browser's address bar), for code signing and for secure messages. The public, as well as state institutions, have been able to use it since Phil Zimmermann published the PGP (Pretty Good Privacy) program in the 1990s. After a run-in with the US Department of Justice, it became generally available.

Secure web-sites caught on rapidly, driven by e-commerce companies' need for secure transactions over the world-wide web. Most e-businesses now use them.

Secure messaging by e-mail did not take off as the originators had hoped. Perhaps people saw no real need for it, or were waiting for others to adopt it first so that they would have someone else to talk to. Development of cryptography support for secure e-mail slowed down; it was still based largely on the 1990s versions and has been criticised, e.g. by Green and Marlinspike[‡], for not keeping up-to-date with changing technology and expectations. Complaints include:

- clumsy key management and does not support *forward secrecy;*
- lack of automatic key synchronisation across devices, so user has to copy them manually;
- obscure trust model so authentication is often not done properly;
- difficulty of adding desired features without a total redesign; and
- complex and old-fashioned *user interface*, requiring either training or recourse to bulky and unreadable documentation.

† See Davis: *Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML.*

‡ See, for example, Matthew Green: *What's the Matter with PGP?* and Moxie Marlinspike: Blog >> GPG and Me.

Cryptography in the Age of Social Media

A revival of consumer demand came from the world of social media. Users wanted secure communication, to keep their activities secret from snoopers or from the police or security services; they wanted it on mobile devices; and they wanted to use it as an appliance, not needing a high degree of IT skills.

To meet this demand, new cryptographic applications (see Appendix III of the *Implementing Cryptography* paper) were developed to make secure communication easier to use and more modern in style by combining:

- instant messaging etc. over the internet – familiar to Social Media users – and other features that users want; and
- the Public Key Cryptosystem – as described earlier, but with a modernised user interface and extended cryptographic support.

Some of these applications provide robust cryptography suitable for serious business and professional users, for whom high security is a priority; others are aimed at the recreational market and provide popular add-ons but less emphasis on security.

Each application comes as part of a proprietary *platform;* to use it, you sign up for an account, which lets you communicate with other users of that platform. You can extend your circle of contacts by opening accounts on several platforms. This is less open than e-mail, which let you communicate freely with anyone else, anywhere on the internet, using any e-mail software.

Other possible drawbacks are that instant messaging lacks the quoting and threading features of e-mail and could make document control more difficult.

Multi-device Support

Many users now own more than one device – for example desktop computer, laptop computer, tablet computer or mobile phone. They may use them in different places and at different times – for instance send an enquiry from a desktop computer at work, receive a reply by mobile phone on the train and send a follow-up query from a laptop at home. They want this conversation to be as seamless as if they were conducting it from a single device in a single place.

To do this, each device's key store must have a copy of your own Private and Public keys and your correspondents' Public keys. In early Public Key Cryptography, you had to propagate them manually from device to device but this was tricky. The new applications synchronise keys between devices automatically.

Forward Secrecy

Forward Secrecy can protect you against leakage of commercially sensitive, compromising or embarrassing information, that you probably did not know – or had forgotten – was on your computer. The biggest risk is often from working copies of data, that computers make without your knowing as it passes through the system; tracking down and purging all such copies is next to impossible.

The cryptography software does not attempt to purge this data; instead, it keeps each item of data encrypted with its own one-off or short-term key. Once the key has expired, no-one can ever decrypt the data again.

In an online conversation with forward secrecy, the software generates a key for each successive message. Even if an attacker cracks the key for one message, previous messages remain secure.

Forward secrecy is not a water-tight guarantee of security. If the sender's device has been infected with malicious software that can spy on the information before it has been encrypted, or the recipient's device with software that can stash away a copy of the information after it has been decrypted, the attacker can still obtain a copy of it; and there is no way to stop the recipient taking a photograph of their screen while the information is on display.

Cryptographic Support for New Application Features

To support these features, applications need extensions to the basic cryptography described in the Anatomy of the Public Key Cryptosystem section. You still have a key, providing decryption and encryption algorithms, but it is created automatically by the software when you sign up to the platform:

– the private key is secret to you and never leaves the device on which it was created; and

– the public key has to be available to your contacts on the platform. Your software publishes it to them via a key store, which may be centralised or distributed according to the application.

But the new features also require short-term dynamic keys. To implement Forward Secrecy, for example, you need a key for encrypting each message. This is implemented in the Signal protocol[*].

[*] See the video presentation at: https://www.youtube.com/watch?v=7WnwSovjYMs. The software combines the Public Key that authenticates the message with a one-time 'pre-key'.

In addition, the user interface has been modernised by a method of creating and managing keys behind the scenes, more in keeping with the social media model (see *Implementing Cryptography* section TOFU).

The Future of Cryptography

Cryptography has always been at risk of keys being cracked, so making the current cryptographic protocols useless. This is a particular concern now because of the development of powerful Quantum Computers.

The widely used RSA mechanisms could become vulnerable to cracking but this is impracticable by classical methods. However it is feasible – though difficult and expensive – using quantum methods, that could be within the reach of large organisations such as governments. A defence[†] has been proposed but, while cheap compared with the attack, it is beyond the reach of normal users.

There is also active research into new communication techniques, that exploit the quantum characteristics of novel materials and so provide security that would be very hard for an attacker to defeat.

Conclusions

1. The theory behind the public key architecture is sound. The method is in wide use and, so far as known, has not yet been cracked. There is a risk that it may be rendered useless by advances in code-cracking. It is hoped that cryptographers will continue to keep ahead of the crackers.

2. The social media based applications have a superb solution for multi-device usage, partly cure the usability difficulties of the older software (though perhaps creating some new ones) and support forward secrecy.

3. However, users may need to sign up for several different platforms and install the appropriate software for each, in order to communicate widely on the internet; and the social media applications do not, at present, cater for some likely needs of business users – for example, integrated e-mail support (or perhaps something better).

Richard Stonehouse

† See Bernstein, Heninger, Lou & Valenta: *Post-quantum RSA*

Appendix – Key Usage in Public Key Cryptography

The Basic Rules

Private and public key work together according to the rules:

1. you do not share your private key with anyone. You may share your public key, because no-one can derive the associated private key from it; and

2. whichever key was used to encrypt something, it can only be decrypted by the **other** key of the **same** key pair.

Key Usage

The private and public keys of the sender and the recipient are used as shown in the table below. Decryption, including the special case of decrypting plaintext, can only be done by the owner of the relevant private (decryption) key. Anyone can encrypt plaintext by using the relevant public (encryption) key; these keys are distributed generally. Decrypted plaintext can be re-encrypted using the public key counterpart of the private key that was used for decrypting it.

Encryption/Decryption	**Signing/Verification**
Sender Side	
Message is written by sender and encrypted by sender using **copy** of **recipient's** public key.	Plaintext signature is created by sender and **decrypted** by sender using sender's own private key.
Recipient Side	
Message is decrypted by recipient using recipient's own private key.	Plaintext signature is re-encrypted by recipient using **copy** of **sender's** public key.