P

PAILLIER ENCRYPTION AND SIGNATURE SCHEMES

In the spirit of earlier encryption schemes suggested by Goldwasser–Micali, Benaloh, Naccache–Stern, and Okamoto–Uchiyama, Paillier proposed in 1999 a public-key cryptosystem [4] (see public-key cryptograhy) based on the properties of nth powers modulo n^2 where n is an RSA modulus (see modular arithmetic and RSA public key encryption). The original observation is that the function $\mathcal{E}(x, y) = g^x y^n \bmod n^2$ is a one-way *trapdoor permutation* (see trapdoor one-way function) over the group $\mathbb{Z}_n \times \mathbb{Z}_n^* \simeq \mathbb{Z}_{n^2}^*$ where the trapdoor information is the factorization of n. The group $\mathbb{Z}_{n^2}^*$ is of order $n\phi$ where $\phi = \phi(n)$ is Euler's totient function of n and the base $g \in \mathbb{Z}_{n^2}^*$ is an element of order $\alpha \cdot n$ for some divisor α of ϕ (for instance $n + 1$ for which $\alpha = 1$). Noting $L(u) = (u - 1)/n$ when $u = 1 \bmod n$, x is recovered from $w = \mathcal{E}(x, y)$ as $x = L(w^\phi \bmod n^2)/L(g^\phi \bmod n^2) \bmod n$ and y is then $(wg^{-x})^{1/n} \bmod n$. When the factorization ϕ of n is unknown, however, recovering x from $\mathcal{E}(x, y)$ and even deciding if $x = 0$ are believed to be hard problems (see computational complexity). This is referred to as the [Decisional] Composite Residuosity assumption ([D]CR for short).

Given the public key (g, n), the Paillier encryption of $m \in [0, n - 1]$ is $c = \mathcal{E}(m, r)$ where r is chosen at random in $[0, n - 1]$. Encryption is therefore probabilistic (see probabilistic public-key encryption) and features homomorphic properties as multiplying $\mathcal{E}(m_1, r_1)$ by $\mathcal{E}(m_2, r_2)$ modulo n^2 provides an encryption of $m_1 + m_2 \bmod n$. This property makes this factoring-based cryptosystem particularly attractive for many cryptographic applications. Paillier encryption is semantically secure (resp. one-way) against chosen-plaintext attacks under the DCR (resp. CR) assumption and was shown to hide $\mathcal{O}(\log n)$ plaintext bits under a slightly stronger assumption [1]. The cryptosystem was extended in several directions. Damgård–Jurik suggested an extension modulo n^s for $s \geq 2$. Variations of different flavors and distributed versions [2,5] of the scheme were introduced (see threshold cryptography). Galbraith showed an embodiment of the scheme on elliptic curves over rings [3].

To sign $m \in 0, n^2 - 1$ under the private key ϕ, one inverts $\mathcal{E}$ to get s_1 and s_2 such that $\mathcal{E}(s_1, s_2) = m$ and the signature is (s_1, s_2). The verification consists in checking if $\mathcal{E}(s_1, s_2) = m$. In virtue of the homomorphic property of $\mathcal{E}$, this provides a blind signature) scheme. As for RSA, a hash function or a padding can be applied before signing to ensure strong security against existential forgery.

Pascal Paillier

References

[1] Catalano, Dario, Rosario Gennaro, and Nick Howgrave-Grahlam (2001). "The bit security of paillier's encryption schemes and its applications." *Advances in Cryptology—EUROCRYPT 2001,* Lecutre Notes in Computer Science, vol. 2045, ed. B. Pfitzmann, Springer-Verlag, Berlin, 229–243.

[2] Catalano, Dario, Rosario Gennaro, Nick Howgrave-Graham, and Phong Q. Nguyen (2001). "Paillier's cryptosystem revisited." *Proceedings of the 8th ACM conference on Computer and Communications Security* ACM Press, New York, 206–214.

[3] Galbraith, Steven D. (2002). "Elliptic curve paillier schemes." *Journal of Cryptology*, 15 (2), 129–138.

[4] Paillier, Pascal (1999). "Public-key cryptosystems based on composite-degree residuosity classes." *Advances in Cryptology—EUROCRYPT'99*, Lecture Notes in Computer Science, vol. 1592, ed. J. Stern. Springer-Verlag, Berlin, 223–238.

[5] Damgård, Ivan and Mads Jurik (2001). "A generalization, a simplification and some applications of paillier's probabilistic public-key system." *Public Key Cryptography—PKC 2001*, Lecture Notes in Computer Science, vol. 1992, ed. K. Kim. Springer-Verlag, Berlin, 119–136.

PASSWORD

A password is a secret that is presented to a verifier to prove a claim, typically during user authentication. A verifier can determine that the claimant knows the secret by comparing the password with a value he has on store. This can be the secret itself, a unique value computed from the secret using, e.g., a one-way function, or a known document encrypted using the password as a key. Passwords are a simple and convenient authentication mechanism but can only provide limited security. More secure authentication based on

cryptographic protocols and hardware is possible, but these options are typically less convenient for users and more expensive to operate.

As a secret, passwords must be protected. Storing or transmitting a password in plain text is a risk, as is writing it down on a sticky note and attaching it to a screen. While these observations are obvious, they are often disregarded in practice. Passwords are also vulnerable to interception during the actual authentication process if the connection between the terminal and the verifier software is not secure, or if the terminal itself can be manipulated, e.g., to record keystrokes. To prevent attacks on the login process that intercept passwords on the path between the terminal and the verifier, the *Orange Book* security criteria (cf. Security Evaluation Criteria) describe a *Trusted Path*, with which systems can provide secure communication so that users can directly communicate with the secure parts of the system.

A characteristic of passwords is that they are directly entered by users, e.g., at a terminal login, or when accessing protected Web sites. Consequently, passwords can only be composed of characters that can be typed on a keyboard. Typical password mechanisms usually only consider the first n characters of a password, so the number of possible passwords is 95^n (with 95 printable characters). For a standard UNIX system, which considers a maximum of eight characters, this amounts to 6.6×10^{15} different values for passwords that are at least eight characters long. A Windows 2000 (TM) password, e.g., can be up to 127 characters long, which gives maximum of 1.4×10^{251} different combinations. While it is typically not feasible for a casual attacker to try all possible combinations of passwords, such an exhaustive key search is still a possible threat when considering more dedicated and resourceful attackers. With current processing and storage technology, it is possible to precompute and store all possible password values for a sufficiently limited search space.

However, the actual search space for typical passwords is considerably smaller than the theoretical limit because passwords need to be remembered by human users. Because arbitrary combinations of characters with no apparent meaning are hard to remember, requiring the use of such passwords would lead to users writing down their passwords, which induces the obvious risk of exposure. Users therefore typically choose shorter passwords using some form of mnemonics or simply familiar terms. This means that the entropy (cf. information theory) of passwords is low, so passwords are vulnerable to guessing.

A particular kind of password guessing attacks are dictionary attacks. A dictionary attack is carried out by trying candidate passwords from a large dictionary with popular words and terms, such as movie actors, characters from cartoons or literature, animal names, computer science, astrological terms, etc. Because attacks like these are simple and have proven to be very effective [1, 2], passwords are sometimes encrypted based on an additional bit string, a salt, which requires attackers using precomputed dictionaries with encrypted passwords to include 2^n variations of each encrypted password, n being the number of bits used for the salt.

If an attacker can install and run programs on the target host system, password security may also be attacked using a Trojan Horse login program. This program would masquerade as a regular system login screen on a computer terminal and capture and store the passwords entered by unsuspecting users. The program may then print a rejection message to the terminal before terminating and starting the regular login program. The existence of the Trojan Horse is unnoticed by users, who believe they simply mistyped their password during the first login attempt. This attack will not work on systems that provide a Trusted Path because communications via such a path are by definition initiated exclusively by the user, i.e., a login prompt would only appear as a system reaction to a user action, e.g., a keyboard interrupt, as in Windows NT (TM).

"A good password is one that is easily remembered, yet difficult to guess [3]." Because of the importance of choosing "good" passwords for maintaining password security, users should be educated and given guidelines for choosing passwords. The most important ones are summarized as rules of thumb below:

1. Choose long passwords to enlarge the search space for an attacker.
2. Do not choose words that are likely to appear in a dictionary, not even with variations.
3. Do not base passwords on any public information about yourself (birthdate, hobby, children's names), because it may help attackers in guessing.
4. Use the initial letters of the words in a sentence that you can remember, e.g., turn a sentence like "my daughter prefers muffins over cheese cake most of the time" into a password "`mdpmoccmott`". Words created in this manner are unlikely to be found in any dictionary.
5. Insert punctuation characters and digits freely, and capitalize some characters, e.g., turn "`mdpmoccmott`" into "`mD4pMo,cCmott`" to further enlarge the search space.

Three main lines of defense against attacks on password security can be identified:

1. *Reactive*—carry out internal dictionary attacks to check for weak passwords. If a weak password is found the respective account should be blocked until the user chooses a new password.
2. *Proactive*—educate users about the importance of choosing good passwords. Prevent them from choosing weak passwords, i.e., reject passwords that are too short, that are found within a dictionary, or are otherwise considered easy to guess. Proactive password checks can be integrated into the system programs that modify a password so that any new password is checked to be reasonably secure before it is accepted.
3. *Secretive*—protect password files so that not even the encrypted passwords can be obtained, which could otherwise be targets of off-line dictionary attacks. For this reason it is standard practice in modern systems to "shadow" out passwords from the password file, i.e., maintain a separate file that can only be read by system administrators.

In summary, passwords are a compromise between convenience, user acceptance, and cost on the one hand and security on the other. In highly sensitive environments, alternative authentication mechanisms should be considered, e.g., mechanisms based on biometric identification, on external devices (cf. Hardware Security Modules) that, e.g., generate one-time passwords, or on combinations of these mechanisms.

Gerald Brose

References

[1] Feldmeier, D.C. and P.R. Karn (1990). "Unix password security—ten years later." *Advances in Cryptology—CRYPTO'89*, Lecture Notes in Computer Science, vol. 435, ed. G. Brassard. Springer-Verlag, Berlin, 44–63.

[2] Klein, D.V. (1990). "'Forcing the cracker:' A survey of and improvements to password security." *Proc. 2nd USENIX Security Symposium*, 5–14.

[3] Morris, R. and K. Thompson (1979). "Password Security: A case history." *Communications of the ACM*, 22 (11), 594–597.

PAYMENT CARD

A payment card is a magnetic stripe or chip card used in a payment scheme, such as debit and credit schemes, as well as electronic cash schemes like Mondex or Proton. Electronic cash schemes require a chip card, and debit and credit schemes are migrating toward chip card technology.

Peter Landrock

PEM, PRIVACY ENHANCED MAIL

PEM provides a number of security mechanisms for protecting electronic mail transferred over Internet by defining a number of protocol extensions and processing procedures for mail messages following RFC 822 (Request For Comment). The security mechanisms include encryption of mails such that only the intended recipient(s) can read the contents of the mail. Other security mechanisms supported by PEM provide message authentication and integrity as well as digital signatures. Using the latter, nonrepudiation of origin may be achieved.

The security mechanisms provided can be based on symmetric cryptography (see key) or public key cryptography. In the latter case X.509 certificates are used to provide the public keys of the sender and recipient as needed (when mails are signed the public key of the originator is required, and when mails are encrypted the public key of the recipient is required).

PEM is defined in detail in RFC 1421 through 1424. RFC 1421 [2] defines the extensions to RFC 822 mail messages and RFC 1423 [4] specifies the cryptographic algorithms (including formats, parameters, and modes of use) to be used within PEM. RFC 1422 [3] and RfC1424 [5] deal with the key management issues—in particular the use of X.509 certificates.

Torben Pedersen

References

[1] http://www.ietf.org/proceedings/94mar/charters/pem-charter.html

[2] RfC1421: Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures. See http://www.rfc-editor.org/rfc.html

[3] RfC1422: Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management (RFC 1422). See http://www.rfc-editor.org/rfc.html

[4] RfC1423: Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and

Identifiers (RFC 1423). See http://www.rfc-editor.org/rfc.html
[5] RfC1424: Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services (RFC 1424). See http://www.rfc-editor.org/rfc.html

PENETRATION TESTING

Penetration testing is part of a security assessment (e.g., Audit) or certification process (e.g., Common Criteria) with as objective to locate and eliminate security vulnerabilities that could be exploited to gain access to the *security target* (system, device or module) by a potential attacker. In this context, the objective of an attacker is to gain access to the security target by breaking or circumventing its security measures. In other words, an attacker needs only to find one vulnerability to successfully penetrate the security target, while the penetrations testers' objective task is to identify all vulnerabilities. Penetration test constraints as well as constraints for an attacker are time, money, amount of effort and resources. Given these constraints (e.g., resources and time), exploiting a vulnerability and successfully penetrating a security target might be practically infeasible even though it is theoretically possible. Hence, finding a security vulnerability does not imply that it always can be exploited easily. Therefore, the penetration tester's objective and task is much broader and labor intensive than that of an attacker.

When a successful penetration in the security target is possible, then this provides evidence that the current security measures are inadequate and need to be strengthened. In addition, when the penetration went unnoticed by the security target, it shows that the intrusion detection mechanisms do not provide an adequate level of security assurance. On the other hand, when a penetration test is not successful, it does not provide evidence that the security target is secure. In the latter case, the penetration test provides evidence that under the given test conditions, the security target did not show obvious exploitable security flaws.

A penetration test assumes the presence of a *security boundary*, as the test is aimed at penetrating it. The security boundary separates the internal components of the security target from the external components (e.g., outside world), and this separation is enforced by various security measures. In general, a security target is composed of hardware, software and human components. Depending on the human component, security assumptions are made and a penetration test might also include human vulnerabilities (e.g., social engineering).

In general, a penetration test assesses the state of the security target at a given point in time, in contrast to penetration tests based on monitoring techniques (e.g., intrusion detection). Hence, any change in the internal and external environments might lead to new vulnerabilities which would require the execution of a new penetration test. Therefore, in practice certain types of penetration tests based on monitoring techniques analyze security target events on a regular interval or even in real-time, and can search for anomalies that might indicate exploitable vulnerabilities. It should be noted that these penetration tests profiles need to be updated on a regular basis.

A penetration test can be passive (observing) and active (interact). The passive penetration test infers with the existence of vulnerability of the security target in a non-intrusive manner. The *passive penetration test* probes the target boundary and scans for security weaknesses in the target environment, without aiming at gaining access and without adversely affecting its normal operation. *Active penetration testing*, on the other hand, includes tests that are more intrusive by nature. The active penetration test (see also physical attacks) exploits security flaws inherently present in the technology of the target environment, design faults or flaws in the object's parameter configuration.

There are two extreme kinds of penetration tests *Zero-Knowledge Penetration Test (ZKPT)* and *Full-Knowledge Penetration Test (FKPT)*. A ZKPT is based on a black box approach, which assumes zero knowledge or assumptions about the security target. The objective of a ZKPT is to identify how much information about the security target can be gained (i.e., is leaked), how the security target can be modeled, and what vulnerabilities can be identified that could be exploited by potential attackers. A FKPT is based on Kerckhoff's principle [1] (see Maxims), and assumes full knowledge of the internal and external components except from the critical security parameters. Based on this knowledge, the security target is tested for security vulnerabilities that could be exploited by potential attackers. In contrast to the FKPT, ZKPT has the advantage that the tester is not tempted to be biased by the security target design specifications and documentation, and will also explore areas beyond its design. Therefore, ZKPT is often the first phase of a penetration testing followed by an FKPT. Between ZKPT and FKPT several hybrid forms exist, which are

optimized based on the security target and the testing constraints.

Tom Caddy

Reference

[1] Kerckhoff, Auguste (1883). "La cryptographie militaire." *Journal des Sciences Militarires*, 9, 5–38. http://www.cl.cam.ac.uk/users/fapp2/kerckhoffs/

PERFECT FORWARD SECRECY

Perfect forward secrecy (PFS for short) refers to the property of key-exchange protocols in which the exposure of long-term keying material, used in the protocol to negotiate session keys, does not compromise the secrecy of session keys established before the exposure. The most common way to achieve PFS in a key-exchange protocol is by using the Diffie–Hellman key agreement with ephemeral exponents to establish the value of a session key, while confining the use of the long-term keys (such as private signature keys) to the purpose of authenticating the exchange (see authentication). In this case, once a session key is no longer used and is erased from memory then there is no way for the attacker to find this key except by cryptanalyzing the Diffie–Hellman exchange (or other applications that used the session key). In particular, finding the long-term authentication key is of no use in learning the session-key value. One essential element for achieving PFS with the Diffie–Hellman exchange is the use of *ephemeral* exponents which are erased from memory as soon as the exchange is complete. (This should include the erasure of any other information from which the value of these exponents can be derived such as the state of a pseudo-random generator used to compute these exponents.)

The PFS property of authenticated Diffie–Hellman exchanges can be highlighted by contrasting it with other forms of key exchange, such as key-transport protocols, where session keys are transmitted between the peers in the exchange encrypted under long-term public keys (see public key cryptography). In this case, the exposure of the long term secret decryption key will compromise the secrecy of *all* session keys (including those erased from memory) that were exchanged under the corresponding public encryption key. This is a major security threat which, in particular, makes the long-term key extremely attractive for attack.

The property of perfect forward secrecy is especially relevant to scenarios in which the exchanged session keys require secrecy protection beyond their lifetime, such as in the case of session keys used for data encryption. In contrast, applications that use the shared keys only for the sake of authentication may not need PFS in their key-exchange protocol (in most cases, finding an expired shared authentication key is of no value for the attacker).

In principle, any public key encryption scheme can be used to build a key exchange with PFS by using the encryption scheme with ephemeral public and private keys. From a practical point of view this requires that the key generation for the encryption scheme be fast enough. For most applications this disqualifies, for example, the use of ephemeral RSA public key encryption for achieving PFS, since the latter requires the generation of two long prime numbers for each exchange, a relatively costly operation. Currently, most systems that provide PFS are based on the Diffie–Hellman key agreement. These protocols can be found in practice (e.g., in the IKE key-exchange protocol [4] for the IPsec standard) and they have been widely studied in many research papers. Some of the well-known key-exchange protocols that provide PFS are STS [2], ISO-9798 [5], EKE [1], SKEME [6], MQVC [8], and SIGMA [7]. The term "perfect forward secrecy" was first introduced in [3].

Hugo Krawczyk

References

[1] Bellovin, S.M. and M. Merritt (1992). "Encrypted key exchange: Password-based protocols secure against dictionary attacks." *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, May, 72–84.

[2] Diffie, W., P.C. van Oorschot, and M. Wiener (1992). "Authentication and authenticated key exchanges." *Designs, Codes and Cryptography*, 2, 107–125.

[3] Günther, C.G. (1990). "An identity-based key-exchange protocol." *Advances in Cryptology—Eurocrypt'89*, Lecture Notes in Computer Science, vol. 434, eds. J.-J. Quisqueter and J. Vandewalle. Springer-Verlag, Berlin, 29–37.

[4] Harkins, D., and D. Carrel (ed.) (1998). "The Internet Key Exchange (IKE)." *RFC 2409*, November.

[5] ISO/IEC IS 9798-3 (1993). "Entity authentication mechanisms—Part 3: Entity authentication using asymmetric techniques."

[6] Krawczyk, H. (1996). "SKEME: A versatile secure key exchange mechanism for internet." *Proceedings of the 1996 Internet Society Symposium on Network and Distributed System Security*, February, 114–127.

[7] Krawczyk, H. (2003). "SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie–Hellman and its use in the IKE protocols." *Advances in Cryptology—CRYPTO 2003*, Lecture Notes in Computer Science, vol. 2729, ed. D. Boneh. Springer-Verlag, Berlin, 399–424.

[8] Law, L., A. Menezes, M. Qu, J. Solinas, and S. Vanstone (2003). "An efficient protocol for authenticated key agreement." *Designs, Codes and Cryptography*, 28, 211–223.

PERSONAL IDENTIFICATION NUMBER (PIN)

A *Personal Identification Number (PIN)* is a relatively short (typically 4–8 digits) numeric string that is used as a password to authenticate a user to a device such as a smart card, an Automated Teller Machine (ATM), or a mobile phone. Standards addressing the management and security of PINs include ANSI X9.8 and ISO 9564.

Carlisle Adams

Reference

[1] Menezes, A., P.C. van Oorschot, and S. Vanstone (1997). *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL.

PHYSICAL ATTACKS

The term "physical attacks" has two quite different meanings in the field of IT security. The first one describes mechanisms to physically penetrating a rather large perimeter, e.g., overcoming an access control system for a server room. The related techniques, for instance picking locks and bridging fences, are outside the scope of the Encyclopedia. Technologies used for perimeter security involve, for instance, intrusion detection sensors and alarm systems.

In the context of cryptographic implementations, "physical attack" is understood as a term which encompasses all attacks based on physical means against cryptographic devices. Physical attacks are of relevance if an adversary gains physical access to the cryptographic device or its near-by environment, e.g., a smart card.

There are two different objectives which have to be regarded in order to counter physical attacks. The first one aims to prevent the disclosure and/or modification of the internal data (e.g., cryptographic keys and application data). For its realization, tamper resistant and tamper response measures are implemented. Another—weaker—approach focuses on the question of whether or not a cryptographic module has been tampered with. For this, tamper detection characteristics are needed. Note that tamper evidence neither prevent the breaking into the cryptographic boundary nor the disclosure of internal data of the cryptographic module.

The term "cryptographic boundary" [2] defines the physical bounds of the cryptographic device that encloses all relevant security components (hardware and software). There are typically external interfaces to the cryptographic boundary, for instance, lines for data communication and power supply. These lines are generally untrusted as they are controlled externally.

In [1], five attack scenarios which indicate the main areas of physical attacks are defined: penetration, monitoring, manipulation, modification, and substitution.

Penetration: Penetration is an active, invasive attack against the cryptographic module. This includes a breaking into the cryptographic boundary of the module. The aim is to intercept data at the internal communication lines or to read out the memory in order to determine the secret keys stored inside the security module.

Monitoring: Monitoring is a passive, noninvasive attack that leaves the cryptographic boundary intact. This class of attacks makes use of the inherent leakage of the cryptographic module, e.g., by measuring the electromagnetic emanation. TEMPEST investigations and side channel analysis are prominent passive attacks based on monitoring.

Manipulation: Manipulation is a noninvasive attack that leaves the cryptographic boundary intact. The attacks aim to obtain a service in an unintended manner [1], mainly at the logical interface. Manipulating attacks may also include changed environmental conditions. For instance, the cryptographic module might be operated under extreme operating conditions which includes the power supply and the environmental temperature. Noninvasive Fault Attacks belong to this category.

Modification: Modification is an active, invasive attack that includes breaking into the cryptographic boundary of the module. Unlike penetration attacks, the aim is to modify internal connections or the internal memories used.

Substitution: Substitution includes the removal of the cryptographic module which is then substituted by an emulating device with a modified implementation of security functions. The cryptographic boundary is not of primary interest in

this attack. Note that the removed module can be used for a comprehensive analysis of the internal construction.

Examples for security requirements include smart card IC protection profiles used by Common Criteria evaluations [3, 4] and the FIPS 140 security requirements [2]. The latter provides information about the implementation of secure computer systems for the use in unprotected areas.

Kerstin Lemke
Christof Paar

References

[1] ISO 13491-1 (1998). *Banking—Secure cryptographic devices (retail), Part 1: Concepts, requirements and evaluation methods* (1st ed.).
[2] FIPS PUB 140-2 (2002). "Security requirements for cryptographic modules." National Institut of Standards and Technology, available at http://csrc.nist.gov/cryptval/
[3] BSI-PP-0002. "Smartcard IC platform protection profile, v1.0." Available at http://www.bsi.bund.de/cc/pplist/ssvgpp01.pdf
[4] PP/9806. "Smartcard integrated circuit protection profile v2.0." Available at http://www.ssi.gouv.fr/site_documents/PP/PP9806.pdf

PKCS

PKCS, the Public-Key Cryptography Standards, are specifications produced by RSA Laboratories in cooperation with interested developers worldwide in order to promote PKI solutions. The PKCS series are referenced in many formal and de facto standards, including ANSI X9, PKIX, SET, S/MIME, and SSL.

The PKCS series is under constant development. Currently, the list comprises:

PKCS #1: RSA Cryptography Standard (includes former #2 and 4)
PKCS #3: Diffie–Hellman Key Agreement Standard
PKCS #5: Password-Based Cryptography Standard
PKCS #6: Extended-Certificate Syntax Standard
PKCS #7: Cryptographic Message Syntax Standard
PKCS #8: Private-Key Information Syntax Standard
PKCS #9: Selected Attribute Types
PKCS #10: Certification Request Syntax Standard
PKCS #11: Cryptographic Token Interface Standard
PKCS #12: Personal Information Exchange Syntax Standard
PKCS #13: Elliptic Curve Cryptography Standard
PKCS #15: Cryptographic Token Information Format Standard

In addition, there are guidelines for contributions to the PKCS series.

Peter Landrock

PKIX—PUBLIC KEY INFRASTRUCTURE (X.509)

PKIX is a working group under IETF, Internet Engineering Task Force, established in 1995 [1]. The charter of the PKIX working group defines the scope and goals of the working group: to develop Internet standards needed to support an X.509-based PKI. This includes profiling ITU PKI standards and developing new standards related to the use of X.509 certificates on Internet. Most results of the working group are published as Request For Comments (RFC).

X.509 version 3 certificates and version 2 of certificate revocation lists as defined by ITU permit a number of standard extensions as well as privately defined extensions. RFC2459 and RFC3280 profile such certificates and certificate revocation lists by recommending usage of standard extensions and also defines a few new extensions. Other standards profile the use of attribute certificates and qualified certificates, while a schema to support PKIX in LDAP is defined in RFC2559.

An informational standard (RFC2527) describes a framework for the definition of certificate policies and *certificate practice statements*.

A number of protocols related to the use of X.509 certificates on Internet have been defined. Central to the management of certificates is RFC2511, which defines a format for certificate requests. Most notably these requests include the possibility to prove possession of the private key corresponding to the public key to be certified and they provide means for authenticating the requester. These certificate requests are used in RFC2510 and RFC2797, which define management protocols for a number of PKI services including issuing, updating and revoking certificates. The protocols in RFC2510 have a number of options and the standard gives a profile for conforming implementations. RFC2797 gives alternative management protocols based on certificate management syntax (and hence much the same security syntax as in S/MIME).

The Online Certificate Status Protocol (OCSP) defined in RFC2560 provides a mechanism for getting timely information about the status of a certificate. This may not be possible using regularly published certificate revocation lists, as a newly revoked certificate may not be listed in the revocation list held by the relying party. In OCSP a request for the status of a number of certificates is answered with a signed message containing the status of these certificates.

RFC3161 defines a protocol for requesting and getting a secure timestamp. Such timestamps may be used as part of a non-repudiation mechanism by timestamping a signed message, but they can also be used in many other applications—including some that are not related to the use of X.509 certificates. A request for a timestamp does not identify the requester and a timestamp on a particular message is obtained without revealing the contents of the message—only the hash value of the message (see Collision-Resistant Hash Function) is revealed to the time stamping authority. Basically, the timestamp is a signed structure containing the hash value of the message and the time. The timestamp also contains policy information indicating the applicability of the timestamp. If necessary more information can be placed in timestamps using the extension mechanism also used in X.509 certificates.

Torben Pedersen

Reference

[1] http://www.ietf.org/

PLAYFAIR CIPHER

This is digraphic, bipartite *substitution* (see substitutions and permutations) $V_{25} \times V_{25} \to V_{25} \times V_{25}$, using a 5×5 Polybios square and a 'crossing step' $(\alpha\beta)(\gamma\delta) \mapsto (\alpha\delta)(\gamma\beta)$. In the example below, the key Palmerstone has been used to make the Polybios square (see mixed, alphabet).

	1	2	3	4	5
1	P	A	L	M	E
2	R	S	T	O	N
3	B	C	D	F	G
4	H	I	K	Q	U
5	V	W	X	Y	Z

ag $\mapsto$ (12)(35) $\mapsto$ (15)(32) $\mapsto$ EC

If a bigram is found in the same row (or the same column), the 'crossing step' degenerates: it takes the cyclically right neighbor: am $\mapsto$ LE; ae $\mapsto$ LP; aa $\mapsto$ LL (or the neighbor cyclically below: dl $\mapsto$ KT; dx $\mapsto$ KL). Modified rules are common, especially the one concerning the doubles: perhaps one letter of the pair will be omitted or replaced with a null; in cases like 'less seven' this will lead to encrypt 'le s& s& se ve n&', where & is a null.

A modified Playfair uses two Polybios squares, one for the first and one for the second character of the bigrams, e.g.,

A	Y	K	I	H		Y	X	U	H	A	
L	B	M	N	P		T	R	K	B	I	
Q	R	C	O	G		P	M	C	G	S	bu $\mapsto$ KY
Z	X	V	D	S		F	D	L	Q	V	
F	W	U	T	E		E	N	O	W	Z	

In case a bigram is found in the same row or column, rules similar to the ones above are to be applied.

Friedrich L. Bauer

Reference

[1] Bauer, F.L. (1997). "Decrypted secrets." *Methods and Maxims of Cryptology*. Springer-Verlag, Berlin.

PMAC

PMAC is a MAC algorithm designed by Black and Rogaway [1] in 2002. A MAC algorithm is a cryptographic algorithm that computes a complex function of a data string and a secret key; the resulting MAC value is typically appended to the string to protect its authenticity. PMAC is a deterministic MAC algorithm based on a block cipher. In contrast to CBC-MAC, PMAC is fully parallelizable, which means that up to t processors can be used in parallel, where t is the number of n-bit blocks of the message. For serial computations, PMAC is about 8% slower than CBC-MAC. The computation of PMAC does not require the knowledge in advance of the message length.

In the following, the block length and key length of the block cipher will be denoted with n and k, respectively. The encryption with the block cipher E using the key K will be denoted with $E_K(\cdot)$. An n-bit string consisting of zeroes will be denoted with 0^n. The length of a string x in bits is denoted with $|x|$. If $i \geq 1$ is an integer, $\mathtt{ntz}(i)$ is the number of trailing 0-bits in the binary representation of i. For a string x with $|x| < n$, $\mathtt{pad}_n(x)$ is the string of length n obtained by appending to the right a '1' bit followed by $n - |x| - 1$ '0' bits. Considered the finite field $\mathrm{GF}(2^n)$ defined using the irreducible polynomial $p_n(x)$; here $p_n(x)$ is the lexicographically first polynomial, chosen

among the irreducible polynomials of degree n that have a minimum number of non-zero coefficients. For $n = 128$, $p_n(x) = x^{128} + x^7 + x^2 + x + 1$. Denote the string consisting of the rightmost n coefficients (corresponding to x^{n-1} through the constant term) of $p_n(x)$ with $\tilde{p}_n$; for example $\tilde{p}_{128} = 0^{120}10000111$. The operation `multx` *(s)* on an n-bit string considers s as an element in the finite field GF(2^n) and multiplies it by the element corresponding to the monomial x in GF(2^n). It can be computed as follows, where s_{n-1} denotes the leftmost bit of s, and $\ll$ denotes a left shift operation.

$$\texttt{multx}(s) = \begin{cases} s \ll 1 & \text{if } s_{n-1} = 0 \\ (s \ll 1) \oplus \tilde{p}_n & \text{if } s_{n-1} = 1. \end{cases}$$

Similarly, the operation `invx`(s) multiplies the string s by the element corresponding to the monomial x^{-1} in GF(2^n). It can be computed as follows, where $\gg$ denotes a right shift operation:

$$\texttt{invx}(s) = \begin{cases} s \gg 1 & \text{if } s_{n-1} = 0 \\ (s \gg 1) \oplus \tilde{p}_n & \text{if } s_{n-1} = 1. \end{cases}$$

The first step of a PMAC computation precomputes some key-dependent constants. Define $\mu \leftarrow \lceil \log_2 t_{\max} \rceil$ where $t_{\max}$ is the maximum number of n-bit blocks in the message. For a key K, one computes $L \leftarrow E_K(0^n)$ and sets $L(0) \leftarrow L$. Next one defines $L(i) \leftarrow \texttt{multx}(L(i-1))$ for $1 \le i \le \mu$ and $L(-1) \leftarrow \texttt{invx}(L)$.

PMAC can now be defined as follows. Set $t \leftarrow \lceil |x|/n \rceil$; in the special case that t equals 0 set $t \leftarrow 1$. Split the input x (of maximum length $n2^n$ bits) into t n-bit blocks $x_1, x_2, \ldots, x_t$, where the last block may be shorter than n bits. Set $\Delta \leftarrow 0^n$ and $\Sigma \leftarrow 0^n$. Now compute for $1 \le i \le t-1$:

$$\begin{aligned} \Delta &\leftarrow \Delta \oplus L(\texttt{ntz}(i)) \\ H_i &\leftarrow E_K(x_i \oplus \Delta) \\ \Sigma &\leftarrow \Sigma \oplus H_i. \end{aligned}$$

As a final step, set $\Sigma \leftarrow \Sigma \oplus \texttt{pad}_n(x_t)$. Finally

$$H_t = \begin{cases} E_K(\Sigma) & \text{if } |x_t| < n \\ E_K(\Sigma \oplus L(-1)) & \text{if } |x_t| = n. \end{cases}$$

The PMAC value consists of the leftmost m bits of H_t.

The designers of PMAC have proved that PMAC is a secure MAC algorithm if the underlying block cipher is a pseudo-random permutation; the security bounds are meaningful if the total number of input blocks is significantly smaller than $2^{n/2}$. The best attack known on PMAC is a forgery attack based on internal collisions [2]. In early 2004, PMAC has not yet been included in any standards.

B. Preneel

References

[1] Black, J. and P. Rogaway (2000). "CBC-MACs for arbitrary length messages." *Advances in Cryptology—CRYPTO 2000*, Lecture Notes in Computer Science, vol. 1880, ed. M. Bellare. Spring-Verlag, Berlin, 197–215.

[2] Preneel, B. and P.C. van Oorschot (1995). "MDx-MAC and building fast MACs from hash functions." *Advances in Cryptology—CRYPTO'95*, Lecture Notes in Computer Science, vol. 963, ed. D. Coppersmith. Spring-Verlag, Berlin, 1–14.

POLICY

Policy describes how environments dictate the behavior of applications and services. Security policies specify how relevant conditions mandate how, when, and/or to whom access to controlled resources is given. For example, the access rights defined by a UNIX file-system state a policy: read, write, and execute bits define the kinds of operations a user is permitted perform on the files or directories. The policy is *evaluated* by determining whether the user has sufficient permissions to perform the operation at the point at which access is attempted. The policy is *enforced* by the file-system by allowing or preventing the operation.

Security policy has historically been divided into two broad classes: provisioning policy and authorization (or access control) policy. Provisioning policies define how software is configured to meet the requirements of the local environment. As is illustrated by the UNIX file-system policy, authorization policies map entities and resources onto allowable action. The following considers these broad classes in serial.

One can view any kind of software configuration as provisioning policy. That is, any aspect of software behavior configured at run-time is policy. This is relevant to the current discussion where configuration affects how security is provided. For example, one of the central goals of the `ssh` remote access utility [1] is to provide confidentiality over the session. The `ssh` policy states which cryptographic algorithm (e.g., Triple DES, Blowfish) should be used to encrypt session traffic (e.g., as specified by host-local configuration file). The policy (i.e., encryption algorithm) dictates how the goal (i.e., confidentiality) is achieved. In general, the degree to which a user or administrator can influence security is largely dictated by the scope of the system's provisioning policies.

General-purpose policy management services support the creation, storage, and enforcement of policy. Note that while these services can be used

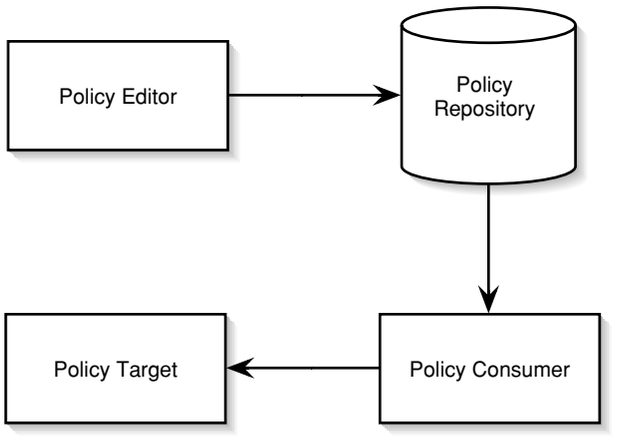

Fig. 1. IETF Policy Framework Working Group architecture—architecture supporting the creation, distribution and enforcement of network management policies

to manage authorization policy, they have historically been targeted to provisioning policy. The IETF Policy Framework Working Group (PWG) has developed a widely adopted reference architecture and lexicon for policy management [2]. This framework defines a collection of components and operations used to manage the network (commonly referred to as *policy-based networking*). Depicted in Figure 1, the architecture defines four logical policy components; *policy editors*, *policy repositories*, *policy consumers*, and *policy targets*.

A policy editor provides interfaces for specifying and validating policy specifications. The policy editor is responsible for detecting (and potentially resolving) inconsistencies in the specification. For example, an `ssh` policy that mandates both *DES* (see Data Encryption Standard) and Triple DES be used for confidentiality is erroneous (only one algorithm should be specified). Such a policy would be flagged as errored, and where available, corrected using a resolution algorithm. The policy editor delivers validated policies to policy repositories as dictated by the environment. A policy repository stores the policies to be used by an administrative domains. The policy repository does not act on or interpret policy.

A policy consumer translates policy into action. The consumer acquires and evaluates the policy relevant to the current environment. The resulting action is communicated to the set of policy targets. Targets enforce policy by performing the actions that implement the defined semantics. For example, again consider the `ssh` policy. A consumer would interpret policy to determine which algorithm is to be used for confidentiality. `ssh` would then act as a policy target by configuring the `ssh` client and subsequently using it to encrypt the traffic (and thus enforce confidentiality).

Authorization policy describes to whom and under what circumstances access to resources is granted. These policies are further defined by an authentication policy and an *access control* policy. The authentication policy states how the identity of the requesting entity must be established. For example, an authentication policy for a UNIX system is the password: the user must provide the appropriate password at the login prompt to be allowed access to the system. How *authentication* is performed is largely defined by environment needs, and outside the scope of this section.

An access control policy maps an identity established during authentication and other information to a set of rights. Rights, often called permissions, defines the types of operations that can be granted, e.g., `read`, `write`, and `execute` on a UNIX file system. The structure and meaning of these policies are defined by their *access control model*. (There are many models, we choose to focus on one.)

One popular model is the *role based* access control model. In this model, the policy defines collections of permissions called *roles* [3]. Users assume roles as they perform different tasks within the system. Hence, the set of rights is strictly defined by the set of rights allowed to the roles they have assumed.

The following example illustrates the use of role-based access control policy. Consider a simplified sealed bid online auction application. Three entities participate in this application: an auctioneer, a bidder, and a seller. The bidder *bids* for goods *sold* by the seller. Once all bids are placed, the auctioneer *clears* the auction by opening the bids and declaring the highest bidder the winner. Once a winner is declared, interested parties can *view* the result. The access control policy for the online auction defines four permissions (described above), bid, sell, clear, and view, and three roles, seller, buyer, and auctioneer. Table 1 describes a policy that assigns the permissions to roles.

The auction policy is enforced at run-time by evaluating the permissions associated with the roles that they have assumed. In any role-based

Table 1. Example role-based policy for online auction application

	Permissions			
Role	Sell	Bid	Clear	View
Seller	Yes	No	No	Yes
Buyer	No	Yes	No	Yes
Auctioneer	No	No	Yes	Yes

system, a user assumes roles through a explicit or implicit software process. The example online auction maps user accounts to roles. For example, the *sally* account is mapped to the seller role. When a user logs into the application as *sally*, she automatically (implicitly) assumes the *seller* role. From that point on, she is free to perform any action allowed by that role. Other accounts are similarly mapped to the *buyer* and *auctioneer* roles, and governed by the associated policy.

One might ask why we do not just assign a user the union of all rights assigned to each role they may assume. Firstly, there may be legitimate reasons that a user be explicitly prohibited from simultaneously assuming more than one role. For example, in the above example, it may be important that a user be prevented from assuming both auctioneer and bidder roles. If a user were to assume both roles, it could "cheat" by viewing all bids before placing its own.

The second reason role-based models are useful is convenience. Roles provide a powerful abstraction for dealing with the potentially many rights that must be managed. In separating the access control from the user, one simplifies the process of evaluating. Moreover, this eliminates the need to alter access control policy each time a entities position changes.

Many access control models have been defined, studied, and ultimately used in real environments. Each model defines a unique way of viewing the relations between protected artifacts, actions, and the entities that manipulate them. Because applications and environments view these concepts in vastly different ways, it is unlikely that any one model will be universally applicable. Hence, like the use policy itself, the selection of a model is a function of taste and system requirements.

Trust Management (TM) systems blur the lines between authorization and provisioning policy. TM policies, frequently called *credentials*, define statements of trust. More specifically, they state that an entity is trusted to access a particular resource under a specified set of conditions. An entity supplies the appropriate set of credentials when accessing a protected resource.[1] If the supplied credentials are authentic and consistent with a *local policy*, the entity is permitted access.

Note that the TM policy alters the traditional policy flow: each entity accessing the resource supplies those policies that are needed to perform an action. This eliminates the need for the policy consumer to discover and acquire policy. It is incumbent upon the user to supply the set of credentials that allow access. Hence, TM systems enable policy in creating widely distributed systems.

```
KeyNote–Version: "2"
Authorizer: "DSA : 4401ff92"     # the Alice CA
Licensees: "DSA : abc991"        # Bob DSA key
Conditions: ((app_domain == "ssh") && (crypt == "3DES"));
Signature: "DSA–SHA1 : 8912aa"
```

Fig. 2. KeyNote credential—specifies a policy stating that Alice allows Bob to connect via SSH if 3DES encryption is used

Provisioning policy is specified in TM systems through the access criteria. Hence, TM systems do not specify provisioning directly, but mandate how the environment provisioning must be provisioned to allow access. This model of policy again departs from traditional uses: the policy infrastructure passively assesses whether the environment is correctly provisioned, rather than actively provisioning it.

The KeyNote system [4] provides a standardized language for expressing trust management policies. KeyNote credentials have three cryptographic components: an *authorizer*, a *licensee*, and a *signature*. The authorizer identifies the authority issuing the policy. KeyNote authorities are cryptographic keys. The signature creates a cryptographically strong (and verifiable) link between the policy and the authorizer. The licensee is to the entity to which the policy refers, e.g., the entity to be allowed access.

KeyNote credentials express a *says* relation: an authority says some entity some aspect has some rights under a set of conditions. To illustrate, the KeyNote credential defined in Figure 2 expresses a policy governing `ssh` access. In this credential, the authorizer authority states that an licensee entity has the right to access the host only if Triple DES is used to implement confidentiality. Note again that both the authorizer and licensee are not really entities, but keys. Hence, any entity that has access to the private key of the licensee, can use this credential to gain access to the host. Moreover, any entity with access to the private key of the authorizer can create and sign such credentials.

On first viewing, the credential in Figure 2 may appear to be ambiguous. The credential does specify which hosts are to be governed. This ambiguity is the source of much of the power of trust management. This credential is only accepted by those systems which have a *local policy* that (directly or indirectly) states that the authorizer has dominion over `ssh` access.

[1] Policies can define access to actions, rather than resource. However, the evaluation of such policies relating to action is identical to resource-oriented policy.

Note that the local policy need not specifically identify the authority in the above credential. The local policy can simply state the authorities it trusts to make policy decisions. Further credentials (typically supplied by the user) that express the delegation by the trusted authorities over `ssh` access are used to construct a delegation chain. Hence, access is granted where a chain of credentials beginning at the local policy and terminating at the access granting credential can be found.

Computing environments are becoming more fluid. Increasing requirements for software systems to be more open, and at the same time, more secure place unique demands on the supporting infrastructure. New environments and changing threats mandate that the kinds of security provided be reexamined and reconfigured constantly. Consequently, systems need to be more flexible and adaptive. Policy is increasingly used as the means by which correct behavior is defined.

Policy is not a panacea. While significant strides have been made in the construction, distribution and use of policy, many areas require further exploration. For example, we know little about the security implications of enforcing several policies (i.e., composing policies) within the same domain. Future policy systems need to find techniques that identify and mitigate interactions between policies. A more systemic area of investigation is scale: how do we deploy policy systems that are feasible in networks the size of the Internet. It is the answers to these questions, rather than the specifics of policy construction, that will determine the degree to which policy systems will be adopted in the future.

Patrick McDaniel

References

[1] Ylonen, Tatu (1996). "SSH—secure login connections over the internet." *Proceedings of 6th USENIX UNIX Security Symposium*. USENIX Association, San Jose, CA, 37–42.

[2] Internet Engineering Task Force (2002). Policy Framework Working Group. http://www.ietf.org/html.charters/policy-charter.html

[3] Sandhu, Ravi S., Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman (1996). "Role-based access control models." *IEEE Computer*, 29 (2), 38–47.

[4] Blaze, M., J. Feignbaum, J. Ioannidis, and A. Keromytis (1999). "The KeyNote trust management system—version 2." *Internet Engineering Task Force*, RFC 2704.

POLYBIOS SQUARE ENCRYPTION

This is a monographic, bipartite *substitution* (see substitutions and permutations) $V_{25} \longrightarrow W_5^2$, known in antiquity (Polybios) for the Greek alphabet. In a modern form, a standard alphabet Z_{25} (left) or a mixed alphabet V_{25} (right) is inscribed into a 5×5 checkerboard (Polybios square), like

	1	2	3	4	5
1	a	b	c	d	e
2	f	g	h	i	k
3	l	m	n	o	p
4	q	r	s	t	u
5	v	w	x	y	z

or

	1	2	3	4	5
1	p	a	l	m	e
2	r	s	t	o	n
3	b	c	d	f	g
4	h	i	k	q	u
5	v	w	x	y	z

An encryption example is given by: heaven $\mapsto$ 23151151121533 resp. heaven $\mapsto$ 411512511525

Friedrich L. Bauer

Reference

[1] Bauer, F.L. (1997). "Decrypted secrets." *Methods and Maxims of Cryptology*. Springer-Verlag, Berlin.

POLYNOMIAL TIME

A *polynomial-time* algorithm is one whose running time grows as a polynomial function of the size of its input. Let x denote the length of the input to the algorithm (typically in bits, but other measures are sometimes used). Let $T(x)$ denote the running time of the algorithm on inputs of length x. Then the algorithm is polynomial-time if the running time satisfies

$$T(x) \leq c_d x^d + c_{d-1} x^{d-1} + \cdots + c_1 x + c_0$$

for all sufficiently large x, where the degree $d \geq 0$ and the coefficients $c_d, \ldots, c_0$ are constants. It is easy to see that the running time will also satisfy the simpler bound

$$T(x) \leq c'_d x^d$$

for a (possibly) larger constant c'_d, and (possibly) larger x. In O-notation, this would be written $T(x) = O(x^d)$.

The term "polynomial" comes from the fact that the bound has same general form as a polynomial in x. The definition of polynomial-time is more general, however; in particular, it does not require that d be an integer.

The definition just given is somewhat informal; for instance, it assumes that the algorithm takes

the same time for all inputs of a given size. If the time varies among inputs of a given size, then $T(x)$ should be an upper bound on the running time for that size. Moreover, if the algorithm involves randomization, then $T(x)$ should be an upper bound on the expected running time of the algorithm (given random choices of the algorithm) for inputs of size x.

As noted above, x refers to the length of the input. However, sometimes it is convenient just to let x be the input itself, as for example when expressing the time it takes to find a prime of length x bits, where x itself is the input to the prime-finding algorithm. (Alternatively, as is standard in complexity theory, one could redefine the input to the prime-finding algorithm as a string of x ones.)

The distinction between polynomial-time algorithms and those with higher complexity such as subexponential time and exponential time is a major concern in computational complexity theory, but in practical cryptography, the primary issue is how fast an algorithm runs for practical-size inputs. For instance, while a polynomial-time algorithm will be faster for all sufficiently large inputs than one that is not polynomial-time, but the crossover-point may be well beyond practical sizes. Likewise, the existence of a polynomial-time algorithm to solve some supposedly hard problem (e.g., integer factoring) would not necessarily compromise security, if the polynomial-time algorithm turned out to have a very large degree d. The search for polynomial-time algorithms is nevertheless an important one in cryptography, as it generally provides for scalability in a system design, since key sizes can be increased with only a moderate effect on running time. Likewise, the lack of polynomial-time algorithms for hard problems provides assurance that a moderate increase in key sizes can have a significant effect on security, which helps to stay ahead of improvements in computing power (see Moore's Law).

A polynomial-time algorithm technically belongs to the higher complexity classes, since each complexity class only expresses an upper bound on the running time, not a lower bound. However, in typical discussions on cryptography (see, e.g., L-notation), the term "subexponential time" excludes algorithms that are polynomial time, and likewise "exponential time" excludes subexponential time and polynomial time.

In addition to running time, the "polynomial" metric is sometimes applied to other measures of complexity in cryptography, such as the memory requirements of an algorithm or the ratio of difficulty between two problems. In general, a *polynomial function*, in this context, is one that is bounded asymptotically by a constant power of its input. In these cases, one may also consider degrees $d < 1$ (which would does not make sense for an algorithm, since it takes $O(x)$ time just to *read* the input). A function that is $O(\sqrt{x})$, or that is $O(1)$—a constant—is technically polynomial in x. If $d < 1$ then the function of x is sometimes called a *small polynomial*. If the function grows more slowly than x^d for every degree $d > 0$, then the function is called *subpolynomial*. An example of a subpolynomial function is $O(\log x)$.

Burt Kaliski

PORTA ENCRYPTION

This particular encryption method works on an alphabet of $N = 2 \cdot \nu$ letters. A Porta encryption step (Giambattista della Porta, 1563) is a simple *substitution* (see substitutions and permutations) consisting of ν swaps (cycles of length 2). Typically, there are ν such swaps, each one designated by *two* key letters in a polyphonic way. A Porta encryption is self-reciprocal. For an example of a PORTA encryption see the section "autokey" in the entry key.

A *Porta table* for Z_{20} (G.B.della Porta and M. Argenti, 1589) is given by:

A B	(am)	(bn)	(co)	(dp)	(eq)	(fr)	(gs)	(ht)	(iu)	(lx)
C D	(ax)	(bm)	(cn)	(do)	(ep)	(fq)	(gr)	(hs)	(it)	(lu)
E F	(au)	(bx)	(cm)	(dn)	(eo)	(fp)	(gq)	(hr)	(is)	(lt)
G H	(at)	(bu)	(cx)	(dm)	(en)	(fo)	(gp)	(hq)	(ir)	(ls)
I L	(as)	(bt)	(cu)	(dx)	(em)	(fn)	(go)	(hp)	(iq)	(lr)
M N	(ar)	(bs)	(ct)	(du)	(ex)	(fm)	(gn)	(ho)	(ip)	(lq)
O P	(aq)	(br)	(cs)	(dt)	(eu)	(fx)	(gm)	(hn)	(io)	(lp)
Q R	(ap)	(bq)	(cr)	(ds)	(et)	(fu)	(gx)	(hm)	(in)	(lo)
S T	(ao)	(bp)	(cq)	(dr)	(es)	(ft)	(gu)	(hx)	(im)	(ln)
U X	(an)	(bo)	(cp)	(dq)	(er)	(fs)	(gt)	(hu)	(ix)	(lm)

Friedrich L. Bauer

Reference

[1] Bauer, F.L. (1997). "Decrypted secrets." *Methods and Maxims of Cryptology*. Springer-Verlag, Berlin.

PREIMAGE RESISTANCE

Preimage resistance is the property of a hash function that it is hard to invert, that is, given an element in the range of a hash function, it should be computationally infeasible to find an input that maps to that element. This property corresponds

to one-wayness, which is typically used for functions with input and output domain of similar size (see one-way function). A minimal requirement for a hash function to be preimage resistant is that the length of its result should be at least 80 bits (in 2004). Preimage resistance needs to be distinguished from two other properties of hash functions: second preimage resistance and collision resistance. A hash function is said to be a one-way hash function (OWHF) if it is both preimage resistant and second preimage resistant. A natural question is to investigate how these concepts are related; it turns out that the answer is rather subtle and requires a formalization [8]. A simplification is that under certain conditions, preimage resistance is implied by the two other properties.

An informal definition is not fully satisfactory, as one can always apply the function to a small set of inputs and store the result; for this set of hash results, it is easy to find a preimage. In order to write a formal definition, one needs to specify according to which distribution the element in the range is selected (e.g., one could choose an element uniformly in the domain and apply the hash function, or one could choose an element uniformly in the range), and one needs to express the probability of finding a preimage for this element. Moreover, one often introduces a class of functions indexed by a public parameter, which is called a key. One could then distinguish between three cases: the probability can be taken over the random choice of elements in the range, over the random choice of the parameter, or over both simultaneously. As most practical hash functions have a fixed specification, the first approach is more relevant to applications.

The definition of a one-way function was given in 1976 by Diffie and Hellman [2]. Preimage resistance of hash functions has been studied in [1, 3–7, 9, 10]. For a complete formalization and a discussion of the relation between the variants and between hash functions properties, the reader is referred to Rogaway and Shrimpton [8].

B. Preneel

References

[1] Damgård, I.B. (1990). "A design principle for hash functions." *Advances in Cryptology—CRYPTO'89*, Lecture Notes in Computer Science, vol. 435, ed. G. Brassard. Springer-Verlag, Berlin, 416–427.

[2] Diffie, W. and M.E. Hellman (1976). "New directions in cryptography." *IEEE Trans. on Information Theory*, IT-22 (6), 644–654.

[3] Gibson, J.K. (1990). "Some comments on Damgård's hashing principle." *Electronics Letters*, 26 (15), 1178–1179.

[4] Merkle, R. (1979). "*Secrecy, Authentication, and Public Key Systems*." UMI Research Press.

[5] Preneel, B. (1993). "Analysis and Design of Cryptographic Hash Functions." *Doctoral Dissertation*, Katholieke Universiteit Leuven.

[6] Preneel, B. (1999). "The state of cryptographic hash functions." *Lectures on Data Security*, Lecture Notes in Computer Science, vol. 1561, ed. I. Damgård. Springer-Verlag, Berlin, 158–182.

[7] Rabin, M.O. (1978). "Digitalized signatures." *Foundations of Secure Computation*, eds. R. Lipton, R. DeMillo. Academic Press, New York, 155–166.

[8] Rogaway, P. and T. Shrimpton (2004)."Cryptographic hash function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance." *Fast Software Encryption*, Lecture Notes in Computer Science, vol. 3017, eds. B.K. Roy and W. Meier. Springer-Verlag, Berlin.

[9] Stinson, D. (2001). "Some observations on the theory of cryptographic hash functions." Technical Report 2001/020, University of Waterloo.

[10] Zheng, Y., T. Matsumoto, and H. Imai (1990). "Connections between several versions of one-way hash functions." *Proceedings of SCIS90, The 1990 Symposium on Cryptography and Information Security*, Nihondaira, Japan, January 31–February 2.

PRETTY GOOD PRIVACY (PGP)

Pretty Good Privacy (PGP) is the most widely used software package for email and file protection. It was primarily developed by Philip R. Zimmermann in the early 1990s and allows to encrypt and digitally sign email messages, individual files or protect complete file systems. Over a decade, PGP advanced from a niche market product of the cryptography community to a mainstream application with easy-to-use user interface. Today, it is available on all major operating systems and integrates as a plug-in to email systems, such as Lotus Notes, Microsoft Outlook, Novell GroupWise and the Eudora mail system. PGP brings its own key management applications and key server, but also interoperates with X.509-compliant public key infrastructures (PKI) and LDAP-based directory services. The current FIPS 140-1 compliant products support SmartCards and PKCS#11-compliant cryptographic devices. Beside the commercial PGP, which is available in a variety of product editions with customized features, a culture of compliant, free software packages has evolved. Partly they stem

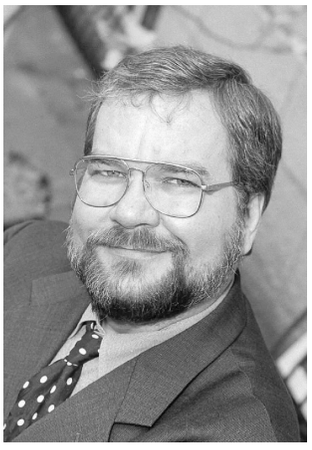

Fig. 1. Phil Zimmermann

from the original PGP source code, published as non-commercial, international versions. Others, like GNU Privacy Guard (GnuPG), have been founded on the OpenPGP standard [14] and were developed independently.

The creator of PGP, Philip Zimmermann, describes himself as a privacy and civil rights activist. He intended to create a public-key encryption software that would support every PC user to privately exchange electronic mail. The actual development of PGP was motivated by the increasing power and efficiency with which U.S. law-enforcement and other governmental agencies could analyze huge amounts of electronic communication, and the emerging U.S. Senate Bill 266, 1991, which would have forced manufacturers to add back doors to secure communication products, permitting governmental authorities access to all conveyed information in decoded form. Senate Bill 266 was later discarded after vigorous protests of libertarian and industry groups [11, 12].

HISTORY: The first version of PGP for Microsoft DOS was released in June 1991 as an Internet give-away through a friend of Zimmermann. It spread rapidly world-wide [6], but faced several problems: Firstly, the novel symmetric cipher algorithm *Bass-O-Matic* turned out to be fundamentally flawed. Moreover, the patented RSA algorithm was implemented without a valid U.S. licence. Earlier, in 1991, Zimmermann had tried to obtain a free licence from RSA Data Security, Inc., but without success, probably because PGP resembled too closely their own product 'Mailsafe'. He decided not to buy a commercial license in order to keep a door open to later distribute PGP as shareware. After all, because the software utilized strong cryptography, PGP fell under the U.S. jurisdiction of the International Traffic in Arms Regulations (ITAR) and Arms Export Control Act (AECA). Therefore, it was illegal to export it from the U.S. to other countries, for instance, through the Internet or Bulletin Board Systems (BBS).

The U.S. Customs Bureau interviewed Zimmermann and others involved in the distribution, caused by complaint from RSA Data Security; documents relating to PGP were subpoenaed from Zimmermann. In addition, a grand jury started gathering evidence, whether to indict Zimmermann on unlawful arms-export. The case raised a controversial public debate on privacy and government-controlled cryptography politics. Zimmermann became a symbol of the civil rights and privacy movement, while facing an up to 5 years imprisonment sentence and huge legal expenses. The investigations lasted from 1993 to 1996. Eventually, the prosecution was declined, at least partially impelled by the public opposition [3, 4].

Version 2.0 released in September 1992, replaced Bass-O-Matic by the Swiss International Data Encryption Algorithm (IDEA) as its principal block cipher, which was licence free for non-commercial products. This PGP version was developed by an informal team of international programmers including Phil Zimmermann, Branko Lankester (Netherlands) and Peter Gutmann (New Zealand) and published its results outside the United States.

Zimmermann sold the exclusive rights for commercial versions to ViaCrypt, which released ViaCrypt PGP 2.4 shortly after the purchase in November 1993 [6]. ViaCrypt resolved the RSA and IDEA patent infringements. Several major and minor ViaCrypt versions leaded up to PGP 4.5.

In spring 1994, the Massachusetts Institute of Technology (MIT) issued PGP version 2.6 as freeware. This was possible because the MIT negotiated an agreement with RSA Data Security to incorporate the RSAREF toolkit, the freely redistributable reference source code of the RSA algorithm (see RSA public key encryption). As part of the deal, the software contained a signature format incompatibility toward earlier versions, which infringed upon RSA patents. Therefore, users were forced to upgrade to 2.6 or be locked out.

In June 1997, PGP 5.0 was published. The source code was completely re-written for the first time since v1.0. New features included a graphical user interface for Microsoft Windows variants and the Apple Macintosh. The collection of cryptographic algorithms was extended by the block ciphers CAST and triple DES (for session keys), the *Secure Hash Algorithm* (SHA family)

and RIPEMD-160 as additional hash functions and, maybe most importantly, by introducing Diffie–Hellmann Digital Signature Standard (DH/DSS) keys, whose patents had recently expired. One new feature was introduced as Corporate Message Recovery (CMR) or Additional Decryption Key (ADK); it permitted an additional, implicit encryption of message session keys under a corporate escrow key, firmly bound to the users certificate. Numerous critics publicly disapproved the feature as a means of antagonistic surveillance, arguing that the available group addressing feature would solve a company's legitimate interest in message recovery already. Furthermore, a serious implementation flaw was revealed (see below), which demonstrated vulnerabilities against subsequent certificate modifications that would attach functioning ADKs, but remain undetected by the software [13].

1997 ViaCrypt merged with Zimmermanns company Phils Pretty Good Software (PPGS) into PGP Inc. In December of the same year, Network Associates, Inc. (formerly McAfee Associates) acquired the PGP Inc.

Beginning with v5.0i, an official international PGP branch was established outside the US by Ståle Schumacher Ytteborg from Norway. Although cryptographic software in electronic form was still strictly export regulated under US laws, no such restrictions applied to printed material. Therefore, the full PGP source code was printed in an OCR font to a 600-page hard cover by MIT Press and shipped from the US to Europe perfectly legal. Until the US export controls loosened in 1999, all major versions up to v6.5.1i were conveyed in printed form, then scanned electronically to recover the original source code and republished internationally [10].

The product releases from v5.0 up to v7.1 added various utilities and functionality such as disk encryption, a graphical key administration tool and key server, a firewall and a virtual private network (VPN) solution and several adaptations to third-party communication software. Up through v6.5.8, NAI disclosed the program source code for public peer review. PGP Corp. continued this practice beginning with v8.0.

After Zimmermann left NAI in February 2001, NAI declared the PGP business segment for sale in July 2001. In order to rescue the PGP product suite, the PGP Corporation was founded with Zimmermann and the cryptographer Bruce Schneier as members of the technical advisory board. It took over most of the product units and copyrights relating to PGP about a year later in June 2002 and announces the shipping of PGP v8.0 for autumn of that year.

OTHER PRODUCTS/PLATFORMS: An independent implementation based on the OpenPGP standard [14], GNU Privacy Guard or GnuPG, was released in version 1.0 in September 1999 by Werner Koch (Germany). Created as a free UNIX command-line utility under the GNU Public License (GPL) in the first place, the software and a large set of auxiliary projects were ported to all major operating systems. GnuPG is functionally complete and OpenPGP-compatible, but avoids patent-protected algorithms, particularly IDEA, in its core distribution [5]. GnuPG was funded in part by the German Federal Ministry of Economics and Technology (BMWi) in an effort to improve and spread secure email exchange. The international PGP [10] and OpenPGP Alliance [9] web sites provide information on further PGP-enabled products.

STANDARDS: Zimmermann compiled an initial specification describing data structures and methods employed in his implementations. Later the PGP2 format was derived and became an Internet Engineering Task Force (IETF) "work in progress" Internet draft, RFC1991, in August 1996 [7]. Since 1998, the specification is further developed as OpenPGP Message Format by the OpenPGP Working Group of the IETF [14]. RFC2440 gives a complete specification of formats and algorithmic methods for secure, electronic communication and file storage. The document intends to provide a framework for interoperable OpenPGP application and defines, in addition to the description of cryptographic techniques, a variety of packet types for message and key items.

WEB OF TRUST: PGP established a decentralized trust model, where each party acts as a user and as a certification authority (CA); all users can be introducers to the Web Of Trust, generate their key pairs, distribute their own public keys, and certify those of other users. As opposed to a centralized, hierarchical, X.509-compliant public key infrastructure (PKI), no standardized or company-specific security policies are applied and no additional information is certified except the mere public key binding to the user's name and email address. PGP users maintain their personal trust relationships locally and rate them by discrete trust values, which mirror the keys validity in the program interaction or user interface. A general framework for trust management structures is presented by Blaze [2].

SECURITY ISSUES: In August 2000, the implementation of the aforementioned CMR/ADK

feature turned out to be flawed. Subpackets of type CMR/ADK are intended to be contained inside an area of the public key packet that is validated with the key pairs self-signature; when located outside, such subpackets need to be ignored for message encryption (and their existence reported as warning to the user). Ralf Senderek inspected several PGP versions up to 6.5.1i and found this condition not satisfied for keys stored under the v4 packet format that was newly introduced with PGP v5.0 [13]. He placed unprotected recovery key information into both *RSA* and *DH/DSS* public key packets and showed that the respective program versions still encrypted messages under both keys. PGP commercial and freeware versions 6.5.8 and later had this bug fixed; later PGP versions offer an option to generate legacy RSA keys, i.e., compliant to the immune v3 format. GnuPG does not support an ADK/CMR feature.

In March 2001, two Czech cryptologists, Klima and Rosa, discovered a weakness in the OpenPGP private keyring specification that may lead to the disclosure of signature keys [8]. Both common packet formats, v3 for *RSA* and v4 for *RSA* and *DH/DSS* keys, are attacked through similar steps, although the respective algorithm parameters are encrypted under a strong, passphrase-derived, symmetric cipher. The stored parameters of the key packets are suitably modified, allowing to recalculate the original private keys, if a message, signed under the tampered key, can be obtained by an attacker. For instance, v3 key packets (only the RSA algorithm is supported) hold the private key parameters enciphered, but their length fields and a 16-bit checksum are stored in the clear. v4 key packets encrypt the length fields and checksum, too. Nevertheless, Klima and Rosa give a prospect of about a half to successfully exploit the CFB mode properties on the last plaintext block of a 1024-bit RSA key packet using the Rijndael/AES block cipher. Regarding DH/DSS v4 key packets, the attack takes advantage of the fact that the public key data fields are not protected, but are nevertheless used during signature generation. Replacing the subgroup generator and a 1024-bit prime parameter with smaller fabricated values reduces the discrete logarithm computation to a feasible problem. In a practical scenario, an attacker needs to gain full access to the victim's private keyring file twice. A first time, so that the keyring can be tampered, and a second time after the victims signature key was recovered, to restore the original keyring to cover up the attack. PGP subjects RSA keys to a built-in integrity check during the signing process, so that altered key parameters are detected, but other products should undergo a comprehensive audit. However, DH/DSS keys of PGP 5.0–7.04 are affected, later versions are fixed, as well as GnuPG beginning with v1.0.7. The Klima/Rosa vulnerability is not highly critical, because the attacker needs to have physical access to the victims computer. Assuming the keyring files cannot be obtained through a network link, which certainly is good practice, it might be more efficient, for example, to replace PGP executables on the target computer, once its location was penetrated, in order to recover the users passphrase directly by monitoring key strokes. On the other side, the format specification does not define a proper, inherent keyring protection.

An implementation flaw in PGP versions 5.0–7.04 allows the execution of malicious code after opening an ASCII armoured file under Microsoft Windows operating systems. Chris Anley (United States) from the security company @stake showed that PGP creates a temporary file of arbitrary names and arbitrary contents on the target machine during the process of parsing the ASCII armour [1]. Assuming the file name was chosen to be an appropriate dynamic link library (DLL) and contains executable code with valid library entry functions, then, due to the Windows DLL loading strategy, the manipulated code might be executed by a subsequent application started. The vulnerability occurs because temporary files are generated in directories such as the current directory, which are also included in the DLL loaders search path and because PGP does not properly remove the temporary file if the parsing fails. NAI supplied patches for PGP 7.03 and 7.04; later releases are not affected anymore.

Clemens Heinrich

References

[1] Anley, Chris (2001). "Windows PGP (Pretty Good Privacy) ASCII armor parser vulnerability." @state security advisory. http://www.atstake.com/research/advisories/2001/a040901-1.txt, April 2001.

[2] Blaze, Matt, Joan Feigenbaum, and Jack Lacy (1996). "Decentralized trust management." *AT&T Research*. Originally published in *Proc. IEEE Conference on Security and Privacy*.

[3] Diffie, Whitfield and Susan Landau (1998). *Privacy on the Line*. MIT Press, Cambridge, MA.

[4] Electronic Frontier Foundation legal cases archive: archive, http://www.eff.org

[5] GnuPG, GNU Privacy Guard web site: http://www.gnupg.org

[6] Garfinkel, Simson (1995). *Pretty Good Privacy*. OReilly & Associates, Inc.

[7] Internet Engineering Task Force, IETF: http://www.ietf.org

[8] Klima, Vlastimil and Tomá Rosa (2001). "Attack on private signature keys of the OpenPGP format." PGPTM programs and other applications compatible with OpenPGP, 2001, http://eprint.iacr.org/2002/076.pdf
[9] OpenPGP Alliance: http://www.openpgp.org
[10] International PGP web site: http://www.pgpi.org
[11] Zimmermann, Philip R. (1996). Testimony of Philip R. Zimmermann to the Subcommittee on Science, Technology and Space, of the U.S. Senate Committee on Commerce, Science, and transportation; http://www.philzimmermann.com/testimony.shtm, 26 June 1996.
[12] Zimmermann, Philip R. (1999). "Why I Wrote PGP", http://www.philzimmermann.com
[13] Senderek, Ralf (2000). *Key Experiments—How PGP deals with Manipulated Keys*, http://senderek.de/security/key-experiments.html, August 2000.
[14] Callas, Jon, Lutz Donnerhacke, and Hal Finney (2002). OpenPGP Message Format. IETF internet draft RFC2440, August 2002.

PRIMALITY PROVING ALGORITHM

A primality test algorithm which, contrary to a probabilistic primality test, always outputs a correct result. See prime number for further discussion and examples of such algorithms.

Anton Stiglic

PRIMALITY TEST

A *primality test* is a criterion that can be used to test whether an integer is prime. The term is also used to designate an algorithm that tests whether a given integer is prime based on some criteria. See also prime number, primality proving algorithm, and probabilistic primality test.

Anton Stiglic

PRIME GENERATION

Different methods can be used to generate primes. The most common use is a primality test. Typically, a random candidate is chosen and tested for primality, if the candidate is found to be composite, another candidate is chosen at random. The process is repeated until a prime is found. One can also test a sequence of integers derived from one randomly chosen candidate. For example, chose a random odd integer n and apply a *sieving procedure* (see sieving) to the sequence $n, n+2, n+4, \ldots, n+2k$, for some k. (With high probability, a prime will be found within $O(\ln n)$ steps.) Then test the sequence incrementaly until the first prime if found. *Probable primes* are generated in this way by using a probabilistic primality test, while a so-called *primality proving algorithm* can be used to generate integers that are guaranteed to be prime. Some primality proving algorithms generate a certificate of primality, which can be used to independently verify the primality of an integer. Another method consists in directly constructing integers that are guaranteed to be prime. These prime generation algorithms are typically recursive; Maurer's method is an example. Primes with certain properties, such as safe primes and strong primes, can also be generated using specialized techniques. For a more detailed discussion on prime generation, and references, see prime number.

Anton Stiglic

PRIME NUMBER

A *prime number* is an integer, greater than 1, whose only divisors (positive integer factors) are 1 and itself. For example, the prime divisors of 10 are 2 and 5. The first 7 primes are 2, 3, 5, 7, 11, 13, and 17. A whole number greater than 1 that is not prime is called a *composite*.

Prime numbers play a central role in number theory (also known as *higher arithmetic*), as can be observed by the *fundamental theorem of arithmetic*, which can be stated as follows: Any positive integer (other than 1) can be written as the product of prime numbers in a unique way (disregarding the order of the factors). Thus, prime numbers can be viewed as the multiplicative building blocks from which all whole numbers are constructed. This is a significant theorem with an analogy to chemistry, saying that just as any natural compound can be broken into basic elements from the periodic table, in a unique way, so can any integer be broken into a product of primes. It is interesting to note that if 1 were considered to be prime, the uniqueness property of the fundamental theorem of arithmetic would not hold. For example, if 1 were a prime, 6 could be written as 2×3, or $1 \times 2 \times 3$. Thus, rather than considering 1 as a prime and losing the uniqueness property, 1 is

considered to be neither a prime nor a composite, instead it is called a *unit*.

Prime numbers play a major role in cryptography, especially in public key cryptography. For instance, in their landmark 1976 paper "New Directions in Cryptography", Diffie and Hellman present a secure key agreement protocol that can be carried out over public communication channels (see Diffie–Hellman key agreement). Their protocol gave birth to the notion of public-key cryptography. The operations in the protocol they describe are based on arithmetic modulo a large prime number (more precisely, the arithmetic was done in a Galois Field of a large prime order, see modular arithmetic and finite field). Since then, many other cryptographic systems that were proposed were based on the use of large prime numbers and mathematical results surrounding these. For example, large prime numbers are used in public-key encryption schemes, key agreement, digital signature schemes, and pseudo-random number generators.

One basic question that can be asked about prime integers is—how may of them exist? The answer is that there are infinitely many of them! The earliest proof of the infinitude of primes is due to the Greek mathematician Euclid (300 B.C.) and can be found in his work, the *Elements*. A modified version of Euclid's proof is: Suppose that there is a finite set of prime integers, say $\{p_1, p_2, \ldots, p_k\}$ for some fixed value k. Now, consider $N = (p_1 \cdot p_2 \cdots p_k) + 1$ (the product can be computed since there are only a finite number of terms). N is not equal to 1 and is greater than any prime in our hypothetical set of finite primes, thus N is a composite and should be divisible by a prime. However, for any prime p_i in our finite set or primes, p_i does not divide N since p_i divides $p_1 \cdot p_2 \cdots p_n$ so it cannot divide $p_1 \cdot p_2 \cdots p_i + 1$ (this is due to a basic theorem in number theory that states that if an integer greater than 1 divides an integer x, it cannot possibly divide $x + 1$ as well). Thus, we have a contradiction! Since the only unfounded assumption we made is about the finite size of the set of primes, we conclude that this assumption is erroneous, and that the number of primes must be infinite.

Not only do we know that there is an infinity of primes, we also know that in any sufficiently large interval there are a good number of them and we can approximate the quantity. This is important, since even though we proved that there is an infinity of primes, we might wonder what is the chance of a large, randomly chosen number being prime?

Let $\pi(x)$ denote the number of primes smaller or equal to x. It can be shown that for any $x \geq 114$,

$$\frac{x}{\ln(x)} < \pi(x) < \frac{5}{4} \cdot \frac{x}{\ln(x)}.$$

This result is, in fact, a refinement of the so-called *Prime Number Theorem*, which states that $\pi(x)$ is *asymptotic* to $x/\ln(x)$. These results allow us to conclude, among other things, that the chance of a random integer x being prime is about $1/\ln(x)$.

Since prime numbers are used in many different algorithms and protocols, the following questions may also be asked: How does one generate a prime integer? And, given an integer, how does one verify whether or not it is prime?

Two techniques exist that answer these questions for small numbers, the *Sieve of Eratosthenes* and *Trial Division*. The sieve of Eratosthenes allows one to generate all the primes less than or equal to a given positive integer n. The technique consists of initially listing all the numbers from 2 up to n in order. For example, to generate all the primes up to 20 we would start with the sequence

2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20.

Starting from 2 (the first prime in the sequence), delete all the multiples of 2 with the exception of 2 itself. In our example, this gives

2, 3, 5, 7, 9, 11, 13, 15, 17, 19.

Next, starting from 3 (the next prime in the sequence), delete all the multiples of 3 (except 3 itself). Our example gives

2, 3, 5, 7, 11, 13, 17, 19.

In general, if the sequence at the tth state is such that p is the tth prime, then the next step is to keep p and delete all the other multiples of p. In our example, the last sequence we came up with is the final sequence, containing all the primes up to 20. This technique is only efficient for small numbers.

Trial division, on the other hand, is a technique that allows us to test the primality of an integer. It is perhaps the simplest primality test one can think of. Given an odd integer n (the only even prime is 2), try to divide it by all odd integers less than n. The number is prime if no divisors are found. It suffices to try dividing it by all odd integers up to $\sqrt{n}$, since if n is composite, at least one prime factor of n must be less than or equal to the square root of n. This primality test is very inefficient, however, since to determine if an integer n is prime, one needs to execute about $\sqrt{n}$ divisions. This is exponential in the size of n and thus not considered to be efficient for large numbers, since

the number of steps is not bounded by a polynomial in the size of n (see polynomial time).

For large integers (such as the ones used in public-key cryptography), we need techniques that are much more efficient for generating and testing primes. A method used in practice to generate large prime numbers is based on the use of an efficient primality test, and can be described as follows. To generate a large prime, execute the following steps:

1. Generate as a candidate an odd random integer n of appropriate size.
2. Test n for primality.
3. If n is determined to be a composite, start again at step 1.

Results regarding the density of prime numbers assure us that we will find a prime number in a reasonable amount of time. As we stated before, the chance of a random integer x being prime is about $1/\ln(x)$, and this chance becomes at least $2/\ln(x)$ if we only test odd integers, since we know that all of the even integers greater than 2 are composite. Thus, if the candidates that are chosen are 2048-bit odd integers, then one can expect to test at most about $\ln(2^{2048})/2$ of them, or about 710 odd integers. Instead of choosing a new random candidate in each iteration of the loop, we could start from a random odd integer candidate n and test the sequence $n, n+2, n+4, \ldots$ until we find a prime. This procedure is called an incremental search. With high probability, a prime will be found within $O(\ln n)$ steps (see O-notation), although there are currently no proofs guaranteeing that a prime will be found quickly for all starting points. Using specific search sequences may allow us to increase the chance that a candidate is prime. For example, to speed up the process, one can take the proposed sequence and apply a *sieving procedure* to remove composites with small prime factors, then test the integers in the resulting sequence. A sieving procedure allows us to eliminate most composites in the sequence rapidly, see sieving. The efficiency of generating large random prime numbers in this way is based on the efficiency of the primality test that is used. A lot of work has been done in primality testing throughout the years. The subject is a very important one, as demonstrated by the following quote, well known among mathematicians, from article 329 of *Disquisitiones Arithmeticae* (1801) by C.F. Gauss:

> The problem of distinguishing prime numbers from composite numbers, and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length ... Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated.

There are two types of primality tests: true and probabilistic. Given a candidate integer, a true primality test will *prove* whether the candidate is prime without any probability of error, while a probabilistic primality test can declare that the candidate is *probably* prime with a certain probability of error. The former can be used to generate so-called *provable primes*, the latter can be used to generate *probable primes*. Algorithms implementing true primality tests that prove the primality of an integer are also called *primality proving algorithms*. Most probabilistic primality tests are correct when they declare a candidate to be composite, but may mistakenly declare a composite to be prime. Typically, when using a probabilistic primality test, step 2 of the prime number generation algorithm, described earlier, is executed multiple times in order to decrease the probability of it falsely declaring a composite to be prime. In what follows, we discuss various probabilistic and true primality testing algorithms.

In the 17th century, Fermat came up with a theorem referred to as Fermat's Little Theorem. The contrapositive of this theorem says that given an integer n, if there exists an integer a, $1 \leq a \leq n-1$ such that $a^{n-1} \not\equiv 1 \bmod n$, then n is guaranteed to be a composite (see modular arithmetic for a discussion of this type of arithmetic). This is the basis of the Fermat primality test. To generate a random prime using Fermat's test with the method we described earlier, do the following:

1. Generate a random integer n of appropriate size.
2. Generate a random integer a, $1 \leq a \leq n-1$ and verify whether or not $a^{n-1} \equiv 1 \bmod n$.
3. If the above equality is true, declare that the number is probably prime, else start again at step 1.

If the chosen candidate is a composite, there is a good chance that the integer a, in step 2, will render the equivalence false, thus proving from Fermat's Little Theorem that n is composite. In step 2, an integer a for which the equivalence is satisfied for a composite n is called a *Fermat liar* (to primality) for n, while such an n is called a *pseudoprime to the base* a (see pseudoprime). For a given candidate, step 2 can be executed multiple times, with different values of a, while the equivalence remains true, in order to increase the

confidence that n is prime. However, the test is not a guarantee of primality, since there exist integers n for which all integers a are Fermat liars. Such n are called *Carmichael numbers*.

In 1976, Miller [24] described a true primality test that determines in polynomial time, assuming the *Extended Riemann Hypothesis* (ERH), whether a given number is prime. ERH is widely believed to be true, but mathematicians have been trying (and failing!) to prove it for over 100 years.

A year later, Solovay and Strassen discovered a randomized algorithm for testing primality based on a criterion due to Euler ([4,31]). The algorithm has a probability of error that can be made arbitrarily small for all inputs. In other words, when the algorithm declares an integer to be prime, there is a small probability that in fact the integer is a composite. This probability of error can be made to be as small as 2^{-100}, or even smaller, while the algorithm still remains efficient. A random value a for which a composite n passes the Euler criteria is called an *Euler liar* (to primality) for n, while such an n is called an *Euler pseudoprime to the base a*. The test resembles Fermat's test, but it doesn't have the drawback of having composites for which every base is a liar, which is one reason why the probability of error can be made arbitrarily small for any input. Furthermore, the result is *unconditional*, meaning that it is not based on any unproven hypothesis (such as ERH).

Rabin [27,28] later modified Miller's algorithm to present it as an *unconditional*, but randomized polynomial-time algorithm, a form that is comparable to the Solovay and Strassen algorithm, with a probability of error that can be made arbitrarily small as well.

This last algorithm (with some optimizations from Knuth [20]) is commonly referred to as the Miller–Rabin probabilistic primality test, or the *strong pseudoprime test*. A random value a for which a composite n passes the primality criteria of the Miller–Rabin test is called a *strong liar* (to primality) for n, while such an n is called a *strong pseudoprime to the base a*. The Miller–Rabin primality test is more efficient then the Solovay–Strassen test and is always correct at least as often (the set of strong liars is a strict subset of the set of Euler liars, which in turn is a strict subset of Fermat liars). The Miller–Rabin test is the most commonly used test for generating primes in practice. One iteration of the Miller–Rabin test will err in declaring a composite integer to be a prime with probability less than $\frac{1}{4}$, while t iterations will err with probability less than $(\frac{1}{4})^t$. That is to say that $\text{prob}[Y_t \mid X] \leq (\frac{1}{4})^t$, where X stands for *n is composite* and Y_t stands for *RepeatRabin(n,t) returned "prime"*, RepeatRabin(n, t) being the algorithm that executes t iterations of Miller–Rabin's test and outputs "composite" as soon as any iteration declares n to be prime, else returns "prime" if each iteration passed. The algorithm is always correct when it declares an integer to be composite. A more interesting result stating that $\text{prob}[X \mid Y_t] \leq (\frac{1}{4})^t$ can also be proven using elementary probability theory and the fact that $\text{prob}[Y_1 \mid X]$ is actually much smaller than $\frac{1}{4}$ [9]. In fact, the error probabilities given are *worst case estimates*. Damgård et al. [16] gave numerical upper bounds for these probabilities which are much smaller when choosing candidates independently from a uniform distribution. So, for example, when looking for a 1000-bit prime, to get a probability of error that is less than 2^{-100}, one can simply choose independently, and from a uniform distribution, 1000-bit numbers and subject each candidate to up to three independent iterations of the Miller–Rabin test, until one is found that passes three consecutive tests. There are also low upper bounds for the probability of error when using an incremental search with Miller–Rabin, these are given by Brandt and Damgård [12].

Other probabilistic algorithms exist that are efficient and have a small probability of error on each round when declaring an integer to be a prime, there are also mixes of tests that seem to be very good.

For example, one can use two iterations of the Miller–Rabin test followed by a single iteration of the Lucas probable prime test to generate 1000-bit primes with probability of error smaller than 2^{-100} [3, 10, 13]. The advantage of this last test is that while composite integers that fool multiple rounds of Miller–Rabin are well known, there is yet no known composite integer that passes even a single Miller–Rabin test to the base 2 followed by a single Lucas probable prime test [26].

The Frobenius–Grantham test [19], on the other hand, is slightly more complicated to implement than the other probabilistic tests mentioned. Furthermore, each iteration takes about three times as long as one iteration of the Miller–Rabin test, although the *worst case* probability of error on each iteration is considerably smaller than that of Miller–Rabin. To achieve an error probability that is less than 2^{-100} for 768-bit candidates or greater, one need only run two iterations of the Frobenius–Grantham test.

Several primality proving algorithms also exist. In 1983, Adleman et al. [2] presented a deterministic primality proving algorithm, whose running time is bounded, for sufficiently large n, by $k \cdot (\ln n)^{c \cdot (\ln \ln \ln n)}$, for some constants k and c, making

it almost polynomial in the size of the input. Cohen and Lenstra [14, 15] simplified the algorithm both theoretically and algorithmically. Further improvements were made by Bosma and van der Hulst [11]. See also [23] for a generalization of the theory used in these tests. The version of this algorithm used in practice is randomized. The algorithm is referred to by different names, such as the Adleman–Pomerance–Rumely or the Cohen–Lenstra–Bosma algorithm, the Jacobi Sum Test, and the Cyclotomy Method. Although efficient in practice (the primality of numbers that are several hundred decimal digits long can be proved in just a few minutes on fast workstations), the algorithm is not easy to program and the possibility of undetected bugs is very likely.

Later, Goldwasser and Kilian [17] proposed a randomized primality proving algorithm, based on elliptic curves, running in *expected* polynomial-time on almost all inputs. This algorithm is inefficient in practice. Adleman and Huang [1] extended these results and designed a primality proving algorithm whose expected running time is polynomial for all inputs. This established, without the use of any unproven assumptions, that the problem of determining whether or not an integer n is prime can be solved, without any probability of error, by a randomized algorithm that runs in time polynomial in the size of the input of the problem. Atkin [5–7] developed a similar algorithm known as Elliptic Curves for Primality Proving (ECPP), which is efficient in practice. The interesting feature of these algorithms is that they produce a certificate of primality, which is a small set of numbers associated with an integer that can be used to prove more efficiently that the integer is prime. So even though the algorithm is also difficult to implement, the results (more specifically the primality certificate) can be verified by an independent implementation, allowing bugs in the code to be detected. ECPP has been used to prove the primality of numbers having more than 1000 decimal digits.

Finally, in 2002, Agrawal et al. [8] discovered a primality testing algorithm that runs in polynomial time in the size of the candidate, for all candidates, without any randomization, no probability of error and not based on any unproven assumptions (such as ERH). This demonstrates that the problem of determining whether or not a given integer is prime is in the complexity class P (see computational complexity). Even though the algorithm is in P, it is far from being as efficient as the probabilistic tests previously mentioned (e.g., Miller–Rabin). In practice, the probabilistic algorithms are preferred for generating prime numbers because of their efficiency and since the likelihood of an error can be made acceptably small, thus conferring no practical advantage to primality proving algorithms for the generation of primes. For instance, the probability that the Miller–Rabin test erroneously declares a composite integer to be prime can efficiently be made smaller than the probability that a computer running a primality proving algorithm would have an undetectable hardware error, leading to an erroneous result.

Other primality proving algorithms exist that are efficient in practice for candidates n having special forms, such as *Mersenne numbers*, or for example when the factorization of $n-1$ is known. A Mersenne number is an integer of the form $2^m - 1$, for $m \geq 2$. A Mersenne number that is a prime is called a *Mersenne prime*. The *Lucas–Lehmer primality test for Mersenne numbers* has been used to prove the primality of integers of over 4 million digits. (Notably, $2^{13466917} - 1$ was proved to be prime using this algorithm.)

The following table summarizes the algorithms that have been discussed. We use the following notation: *Rand* stands for a randomized algorithm, while *Det* stands for a deterministic algorithm. *PT* is for polynomial-time. *Prob* stands for a probabilistic primality test, while *PProv* designates a primality proving algorithm and *PProvCert* a PProv that also produces a certificate of primality. Note that a deterministic polynomial-time primality testing algorithm implicitly provides a trivial certificate of primality consisting simply of the number determined to be prime.

Other techniques do not generate prime numbers by applying a primality test to randomly chosen candidates, rather they construct, in a special way, integers that are guaranteed to be primes. Examples of constructive prime generation algorithms are Shawe–Taylor's algorithm [30] and Maurer's method [21, 22], which are both recursive randomized algorithms that return guaranteed primes. The former will reach roughly 10% of all primes of a specified size, while the latter generates primes that are almost uniformly distributed over the set of all primes of a specified size.

Various standards describe algorithms for generating prime numbers. For example, the sole purpose of the ANSI standard X9.80—2001 [3] is to describe primality tests and prime number generation techniques for public-key cryptography. The standard is intended to be the normative reference in this topic for other ANSI X9 standards. Three probabilistic methods for testing integers for primality are described (Miller–Rabin, Lucas and Frobenius–Grantham), four deterministic

Algorithm name	Type	Based on	Year	Reference
Solovay–Strassen	Prob. Rand, PT	Euler criteria	1977	[4,31]
Miller–Rabin (strong pseudoprime test)	Prob. Rand, PT	Fermat's Little Theorem and the fact that ± 1 are the only square roots of 1 modulo an odd prime	1976–1980	[20,24,27,28]
Adleman–Pomerance–Rumely (Cohen–Lenstra–Bosma, Jacobi Sum Test, Cyclotomic Method)	PProv. Det, almost PT	Set of congruences which are analogues of Fermat's theorem in certain cyclotomic rings	1983	[2,11,14,15]
Goldwasser–Kilian	PProvCert. Expected PT	Elliptic curves	1986–1992	[1,17]
Elliptic Curve Primality (ECPP, Atkin, Atkin–Morain)	PProvCert. Expected PT	Elliptic curves	1986–1993	[5–7]
Frobenius–Grantham	Prob. Rand, PT	Quadratic polynomials and Frobenius automorphism	1998	[19]
Agrawal–Kayal–Saxena	PProvCert. Det, PT	Variation on Fermat's Little Theorem	2002	[8]

methods (including ECPP and trial division, for sufficiently small primes), along with methods for generating prime numbers that use these primality tests, as well as a description of a sieving procedure. The standard also describes two methods for direct construction of prime numbers (Maurer's algorithm and the Shawe–Taylor algorithm). There are also descriptions of methods for generating primes with additional properties, such as safe primes and strong primes. NIST's FIPS 186-2 [25] describes the Miller–Rabin primality test and an algorithm for generating primes with certain properties for the Digital Signature Standard. The algorithm uses the Miller–Rabin test or any other primality test where the probability of a non-prime number passing the test can efficiently be made to be at most 2^{-80}. The algorithm generates two sufficiently large primes, q and p, in such a way that q divides $p-1$. Furthermore, the algorithm provides a *seed* that can be used to demonstrate, under reasonable assumptions, that the primes were not intentionally constructed to be "weak" in a way that the entity who constructed them could subsequently exploit their structure to recover other entities' cryptographic private keys.

Anton Stiglic

References

[1] Adleman, L.M. and M.-D.A. Huang (1992). "Primality Testing and Abelian Varieties over Finite Fields." Lecture Notes in Mathematics, vol. 1512, eds. E. Gimenez and C. Paulin-Mohring. Springer-Verlag, Berlin, Heidelberg, New York.

[2] Adleman, L.M, C. Pomerance, and R.S. Rumely, (1983). "On distinguishing prime numbers from composite numbers." *Annals of Mathematics*, 117, 173–206.

[3] American National Standards for Financial Services (2001). Prime number generation, primality testing, and primality certificates: X9.80.

[4] Atkin, A.O.L. and R.G. Larson (1982). "On a primality test of Solovay and Strassen." *SIAM Journal on Computing*, 11 (4), 789–791.

[5] Atkin, A.O.L. (1986). *Lecture Notes of a Conference*, Boulder, Colorado, August 1986.

[6] Atkin, A.O.L. and F. Morain, (1993). "Elliptic curves and primality proving." *Mathematics of Computation,* 61 (203), 29–68.

[7] Atkin, A.O.L. and F. Morain, (1992). "Finding suitable curves for the elliptic curve method of factorization." *Mathematics of Computation*, 60 (201), 399–405.

[8] Agrawal, M., N. Kayal, and N. Saxena (2002). *PRIMES is in P*.

[9] Beauchemin, P., C. Brassard, C. Crépeau, C. Goutier, C. Pomerance (1998). "The generation of random numbers that are probably prime." *Journal of Cryptology*, 1, 53–64.

[10] Baillie, R.J. and S.S. Wagstaff, Jr. (1980). "Lucas pseudoprimes." *Mathematics of Computation*, 35, 1391–1417.

[11] Bosma, W. and M.-P. van der Hulst (1990). "Faster primality testing." *Advances in Cryptology—EUROCRYPT'89*, Lecture Notes in Computer Science, vol. 434, eds. J.-J Quisquater and J. Vandewalle. Springer-Verlag, Berlin, 10–13, 652–656.

[12] Brandt, J. and I. Damgård (1993). "On generation of probable primes by incremental search."

Advances in Cryptology—CRYPTO'92, Lecture Notes in Computer Science, vol. 740, ed. E.F. Brickell. Springer Verlag, Berlin, 358–370.
[13] Pomerance, C., J.L., Selfridge, and S.S. Wagstaff, Jr. (1980). The pseudoprimes to 25×10^9. *Mathematics of Computation*, 35 (151), 1003–1026.
[14] Cohen, H. and A.K. Lenstra (1986). "Implementation of a new primality test." *Mathematics of Computation*, 48 (177), 103–121, S1-S4.
[15] Cohen, H. and H.W. Lenstra, Jr. (1984). "Primality testing and Jacobi sums." *Mathematics of Computation*, 42 (165), 297–330.
[16] Damgård, I., P., Landrock, and C. Pomerance (1993). "Average case error estimates for the strong probable prime test." *Mathematics of Computation*, 61 (203), 177–194.
[17] Goldwasser, S. and J. Kilian (1986). "Almost all primes can be quickly certified." *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, 316–329.
[18] Gordon, J. (1985). "Strong primes are easy to find." *Advances in Cryptology—EUROCRYPT'84*, Lecture Notes in Computer Science, vol. 209, eds. T. Beth, N. Cot and I. Ingemarsson. Springer-Verlag, Berlin, 216–223.
[19] Grantham, J. (1998). "A probable prime test with high confidence." *Journal of Number Theory*, 72, 32–47.
[20] Knuth, D. (1981). *The Art of Computer Programming—Seminumerical Algorithms*, vol. 2 (2nd. ed.). Addison-Wesley, Reading, MA.
[21] Maurer, U.M. (1995). "Fast generation of prime numbers and secure public-key cryptographic parameters." *Journal of Cryptology*, 8 (3), 123–155.
[22] Maurer, U.M. Fast generation of secure RSA-moduli with almost maximal diversity. *Advances in Cryptology—EUROCRYPT'89*, Lecture Notes in Computer Science, vol. 434, eds. J.-J. Quisquater and J. Vandewalle. Springer-Verlag, Berlin, 636–647.
[23] Mihăilescu, P. (1997). "Cyclotomy of Rings and Primality Testing." *PhD Thesis*, Swiss Federal Institute of Technology Zürich, Diss. ETH No. 12278.
[24] Miller, G.L. (1976). "Reimann's hypothesis and tests for primality." *Journal of Computer and System Sciences*, 13, 300–317.
[25] NIST (1993). "Digital signature standard." FIPS PUB 186, February 1993.
[26] Pomerance, C. (1984). *Are There Counter-Examples to the Baillie-PSW Primality Test*?
[27] Rabin, M.O. (1976). "Probabilistic algorithms." *Algorithms and Complexity: New Directions and Recent Results*, ed. J.F. Traub. Academic Press, New York, 21–39.
[28] Rabin, M.O. (1980). "A probabilistic algorithm for testing primality." *Journal of Number Theory*, 12, 128–138.
[29] Rivest, R.L., and R.D. Silverman (1998). "Are 'strong' primes needed for (RSA)?" Technical Report. RSA Data Security, Inc., Redwood City, CA, USA.
[30] Shawe-Taylor, J. (1986). Generating strong primes. *Electronics Letters*, 22, 875–877.
[31] Solovay, R. and V. Strassen (1977). "A fast Monte-Carlo test for primality." *SIAM Journal on Computing*, 6 (1), 84–85.

PRIMITIVE ELEMENT

A primitive element of a finite field is a root of the field polynomial that is also a generator of the multiplicative group of the field.

Let $f(x)$ be the field polynomial of degree n for an extension field $\mathbf{F}_{q^d}$ constructed over a subfield $\mathbf{F}_q$, and let α be a root of $f(x)$. If α is also a generator of $\mathbf{F}^*_{q^d}$, i.e., if the set of elements

$$\alpha, \alpha^2, \alpha^3, \ldots$$

traverses all elements in $\mathbf{F}^*_{q^d}$, then α is a *primitive element* and *f(x)* is called a *primitive polynomial*.

Not all field polynomials are primitive, but it is often convenient for implementation to use a primitive polynomial. Every finite field $\mathbf{F}_{q^d}$ has $\phi(q^d - 1)$ primitive elements, and $\phi(q^d - 1)/d$ primitive polynomials, where ϕ is Euler's totient function.

Burt Kaliski

PRIVACY

A "name", or *identity*, is a set of information that distinguishes an entity from every other within a given environment. In some environments, the "name" may just be a given name; in other environments, it will be a given name and a family name; in still others, it may include additional data such as a street address, or may be some other form entirely (e.g., an employee number). In all cases, however, the identity depends upon the environment: the size and characteristics of the environment determine the amount of information required for uniqueness.

Regardless of the information tied together to form the identity, however, this "name", or "nym", will be one of three possible types: an *anonym* ("no name"); a pseudonym ("false name"); or a *veronym* ("true name"). These three categories of nyms are defined by the amount of linkage that is possible across transactions. Anonyms are meaningless or null names for one-time use, and so no linkage can be made (1) to the actual entity involved in any given transaction and (2) between transactions. Pseudonyms are either meaningless or

apparently meaningful names for multiple uses that bear no relation to the true name. Thus, no linkage can be made to the actual entity involved in any given transaction, but it is clear that the same (unknown) entity is involved in different transactions. Veronyms are, or very readily disclose, the name of the physical entity in the real world, and so linkage can be made both to the entity involved and across different transactions. Note that linkage to a real-world entity (such as a machine, device, software process, or human person) is central to the notion of *privacy*, but may or may not be relevant to any given application—for many applications, anonyms, pseudonyms, and veronyms are all valid identities for the entities with which they deal.

PRIVACY CONCEPTS: *Privacy* may be defined as an entity's ability to control how, when, and to what extent personal information about it is communicated to others (see Brands [2], p. 20). In order to understand privacy, then it is important to understand what "personal information" is and to understand the ways that personal information can be controlled.

Personal Information

Personal information can include intrinsic physical data such as name, birth date, and gender, but can also include location data (such as home address and telephone number), financial data (such as salary, bank account balance, or credit card number), user-created data (such as confidential correspondence or a list of personal preferences), and data assigned by other entities (such as a bank account number or a social security number). Despite this wide range of types of personal information, it is sometimes useful to distinguish the subset of personal information that reveals the identity of the entity ("identifying information") from all the other types because identifying information can be sensitive on its own, whereas other types of personal information (e.g., salary) are typically sensitive only when revealed in conjunction with identifying information.

Communication of identifying information to unintended parties may be referred to as "exposure": the identity of the entity is exposed to others. Communication of other types of personal information to unintended parties may be referred to as "disclosure": this information has been disclosed to others. Both exposure and disclosure may be direct or indirect. Direct exposure/disclosure is the determination of user identity or other personal information by an observer or by another participant in the exchange from the explicit contents of a single transaction or message. Indirect exposure/disclosure is the determination of user identity or other personal information by inference or from the correlation of the contents of several transactions or messages.

A primary goal of any privacy infrastructure, then, is to provide confidentiality of identifying information and other personal information when desired. That is, the infrastructure must enable an entity to limit the release of this information both directly and indirectly in accordance with the entity's wishes.

Techniques for Control

There are a number of ways in which confidentiality of identifying information may be enabled by a privacy infrastructure. For example, anonymous Web or e-mail services may be provided to entities, or protocols accepting anonymous authentication (proof of ownership of certain attributes without the need to reveal identifying data) may be supported. For environments in which multiple transactions will take place over time or across many servers, the use of pseudonymous identifiers may be supported—either one at a time per entity, or an unlimited number at a time per entity, depending upon the requirements of the environment (see the Trusted Computing Platform Alliance architecture [5] for an example of multiple pseudonym support in a computing environment).

There are two primary techniques for providing confidentiality of non-identifying personal information: encryption and access control. Personal information may be encrypted so that it may only be seen by intended recipients while it is in transit (e.g., as it travels over a Secure Sockets Layer, SSL, channel) or while it is in storage. In storage, it may be encrypted at the application layer for specific entities ("Alice", "Bob", "Charlene") or for particular roles ("Faculty", "Senior Managers", "Hospital Staff"). Encrypting personal information ensures that—at least until it is decrypted—no unintended recipients will be able to read the data.

Access control limits who may do what with a given resource (see authorization architecture). When the resource is personal information, access control allows an entity to explicitly specify which other entities may (or may not) read this information. The effect is similar to encryption (only a restricted set of entities may see the data), but the data itself may not be rendered unintelligible; rather, a Policy Enforcement Point (PEP) enforces

decisions rendered in accordance with access policies created by the subject/owner of the personal information. Any entity not meeting the conditions stipulated by the access policy will be prevented by the PEP from looking at the protected data.

More generally, an entity may wish to control other aspects of its personal data than simply who can read it. In particular, the entity may want to control such things as collection, use, access, dissemination, and retention of its personal information. Access policies and reliable decision and enforcement engines are again the technique for this control. In some environments, however, there are two components of the access policy that must be taken into account. If personal data will be stored on a server or site remote from the entity, then the owner or administrator of that site may create a policy stating what its practices are with respect to privacy (e.g., "We collect and use your name and address information for the purpose of shipping goods to you; we retain this data for a maximum of six months; and we do not sell or in any other way communicate this data to any other party for any reason"). This server policy is one component of the complete access policy. The other component is the policy created by the entity itself, specifying use, retention, and so on, of its data. When access to this data is requested by any entity, both components of the access policy must be consulted and followed by the decision engine. Any conflicts between the two components (e.g., the server says it will retain personal data for 6 months, but the owner/subject of the data specifies a retention period of no longer than 3 months) must be resolved in an acceptable fashion (e.g., the owner/subject policy takes precedence over the site's policy).

Another critical aspect in controlling the communication of personal information to other entities is known as "individual access". This means that the owner/subject of personal data is given read/write access to at least a subset of this data in order to ensure correctness and completeness of the information. This includes access to data created by the entity itself (e.g., address and credit card information), access to data created about the user (e.g., medical examination results), and access to audit information regarding data use and dissemination.

PRIVACY PRINCIPLES: A number of organizations, regulatory bodies, and government agencies have taken a keen interest in privacy issues in recent times. This has resulted in a relatively large collection of privacy guidelines and privacy principles that have been developed in Europe, North America, and around the world. Examples of such guidelines and principles include Bill C-6, the CSA Model Code, the EU Directive on Data Protection, the Health Insurance Portability and Accountability Act (HIPAA), the Gramm–Leach–Bliley Act (GLBA), the OECD Privacy Principles, and the Fair Information Practice Principles.

Although not identical, many of these initiatives do have significant overlap in the issues that they cover. These include topics such as the following: accountability, identifying purposes, user consent, limiting collection of data, limiting use/disclosure/retention of data, accuracy, safeguards, openness, individual access, and compliance. Notions such as user notice and user choice also figure prominently in much of this work.

It may be noted, however, that virtually all of these initiatives are focused on privacy principles that are of interest to the corporate or server entity that holds personal information on behalf of users. There is very little comparable work that is focused on principles that are of specific interest to the users themselves. Although these views will coincide in many ways, the change in perspective can lead to subtle differences or changes in emphasis. User-focused privacy principles deal with the issues that are most important to the owner/subject of personal data and will include the four areas discussed in the previous section: confidentiality of identifying information; confidentiality of other personal information; access to personal data for the purpose of ensuring correctness; and control over the use, retention, dissemination, and so on, of personal information.

PRIVACY TECHNOLOGIES: A number of technologies have been developed over the years to enable and preserve the privacy of entities participating in transactions on electronic networks such as the Internet. Collectively, these have come to be known as Privacy Enhancing Technologies (PETs). This section provides examples of some PETs that are intended to address different portions of the privacy spectrum.

Onion routing (see also MIX networks) and subsequent technologies built on that framework such as *Crowds*, addresses the requirement for anonymity—confidentiality of identifying information—in Web browsing and in messaging (primarily e-mail) environments. The basic idea is that the message from the originator is bounced around a relatively large set of participants in a fairly random fashion before being sent to the eventual destination. The destination, as well as the other participants, may know the immediately preceding hop, but do not know whether there were any hops prior to that one. Thus, the identity

of the originator (including machine identity, IP address, and so on) is kept hidden from all observers. Some technologies allow the possibility of pseudonymous identifiers for participants so that transactions may be linked over time to a single (unknown) entity for continuity, context building, personalization, or similar purposes.

Encryption is commonly used as a technology to provide confidentiality of data, including personal data. As noted above, encrypting data for specific individuals or roles can ensure that unintended parties are not able to read the data while it is in transit or in storage. Symmetric encryption algorithms (see symmetric cryptosystem) are used to scramble the data, with key management/key establishment supported by out-of-band means or by technologies such as Kerberos, Public Key Infrastructure (PKI), or Secure Sockets Layer. Encrypting personal data for a role can be an attractive option when the individuals that need to see that data may vary over time, but it does require a supporting infrastructure in which, at any given time, one entity can reliably determine whether or not another entity is a valid member of that role. A PKI that uses attribute certificates, Security Assertion Markup Language (SAML) assertions, or similar technology for its attribute management (see authorization architecture) is one way to build this supporting infrastructure [1].

Access control is an important technology for restricting the dissemination of personal information. Rules and policies can be established regarding who can do what with this data, and trusted components can make and enforce access decisions on the basis of these policies. Technologies such as SAML and eXtensible Access Control Markup Language (XACML) provide the data structures, protocols, and policy syntax to allow a relatively comprehensive access control infrastructure to be built in the service of privacy. Other technologies such as CORBA and DCE can also provide much of the required functionality in this area.

Finally, technologies that address the control (by both server and entity) of personal information include Platform for Privacy Preferences Project (P3P) and A P3P Preference Exchange Language (APPEL). P3P [3] is a mechanism for Web sites to advertise their privacy policies and practices in a machine-readable way so that a user browsing to a site may automatically be warned if the site's policy does not match his/her own preferences. P3P policies can include statements about the types of personal data that an owner/subject can access, the ways in which the owner/subject can resolve disputes about the site's privacy practices, the purpose(s) for which personal data will be used by the site, other recipients with whom personal data may be shared, and the period for which personal data will be retained by the site.

APPEL [4] is a technology to give users some control over the use of their personal information. With P3P, a user can browse to a site, download its privacy policy, compare this policy with his/her own preferences, and make an informed decision as to whether to continue dealing with this site (e.g., submit personal information or make a purchase). APPEL allows the possibility of some negotiation with the site: after downloading the site's P3P policy and examining it, the owner/subject can respond with his/her own list of preferences (a narrowed-down set of values of various elements in the policy). The site can then choose to treat this user's personal data in accordance with this more restricted P3P policy, or can reject this modified policy. Depending upon the site's response, the user can again decide whether to continue dealing with this site. APPEL gives users a bit more control over how their data is used than P3P alone, which simply informs the user of a site's current practices.

Carlisle Adams

References

[1] Adams, C. and S. Lloyd (2003). *Understanding PKI: Concepts, Standards, and Deployment Considerations* (2nd ed.). Addison-Wesley, Reading, MA.

[2] Brands, S. (2000). *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA.

[3] Cranor, L., M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle (2002). "The platform for privacy preferences 1.0 specification." W3C Recommendation, 16 April 2002; see http:/www.w3.org/TR/P3P

[4] Cranor, L., M. Langheinrich, and M. Marchiori (2002). "A P3P preference exchange language 1.0." W3C Working Draft, 15 April 2002; see http:/www.w3.org/TR/P3P-preferences for current status.

[5] The Trusted Computing Platform Alliance; see http:/www.trustedpc.org/home/home.htm for details.

PRIVILEGE MANAGEMENT

Privilege management is a subset of general "authorization data" management (see authorization architecture) in which the data being managed are *privileges* granted or bestowed upon entities in an environment. A *privilege* may

be defined as follows [1]: "a right or immunity granted as a peculiar benefit, advantage, or favor".

Carlisle Adams

Reference

[1] Merriam-Webster OnLine, http://www.m-w.com/cgi-bin/dictionary

PROBABILISTIC PRIMALITY TEST

A *probabilistic primality test* is a primality test that has a probability of error. Such tests typically pick a random number called a *witness* and verify some criteria involving the witness and the candidate being tested. Most probabilistic primality tests used in practice will not err when declaring an integer to be composite, but have a probability of error when declaring an integer to be prime: they either declare a candidate to be *definitely composite* or *probably prime*. A candidate that passes such a test—whether prime or composite—is called a *probable prime* for that test. A composite number that erroneously passes such a test is called a pseudoprime.

See prime number for further discussion and examples of such tests.

Anton Stiglic

PROBABILISTIC PUBLIC-KEY ENCRYPTION

Probabilistic public-key encryption is a public-key encryption scheme (see public key cryptography) where the ciphertext of the same message under the same public key differs on every run of a probabilistic Turing machine. That is, a random coin toss of the Turing machine is used in the encryption algorithm. The notion was proposed in contrast to the RSA public-key encryption scheme, which is deterministic in the sense that the ciphertext is always fixed given a public key and a plaintext.

The early date examples of probabilistic public key encryption schemes are the Goldwasser Micali encryption *scheme* and the ElGamal public-key encryption scheme, and many others followed. It is known that an encryption scheme that satisfies provable security such as semantic security must be probabilistic. Since the original RSA encryption scheme was not probabilistic, several padding techniques such as OAEP are considered so that the padded RSA scheme becomes probabilistic and satisfies security properties, such as semantic security.

Kazue Sako

Reference

[1] Goldwasser, S. and S. Micali (1984). "Probabilistic encryption." *Journal Computer and System Sciences*, 28, 270–299.

PRODUCT CIPHER, SUPERENCRYPTION

Product ciphers are ciphers that are built as a composition of several different functions. In a special case when all the functions are the same, the cipher is called *iterative cipher* and the functions are called *rounds*. The intuition behind such constructions is inspired by the analogy with mixing transformations studied in the theory of Dynamical Systems, which was first noted by Shannon [3]. The IBM team followed this approach in the design of *Lucifer* and the Data Encryption Standard (DES). Rounds of iterative ciphers are typically keyed with different *subkeys* or at least involve different round constants to break self-similarity, which otherwise would be vulnerable to slide attacks. Note that the individual round transformation might be cryptographically weak. The strength of the whole construction relies on the number of iterations. The choice of the proper *number of rounds* is a difficult task, which is performed via cryptanalysis of the cipher. Most of the modern block-ciphers have an iterative structure for the reasons of compact hardware, software implementation and in order to facilitate the analysis. The typical number of rounds in modern ciphers ranges between 8 and 32. By their structure, iterative block ciphers may be divided into two large classes: Feistel ciphers and substitution–permutation networks (SPN).

If one takes the composition of several ciphers one gets what is called multiple encryption or *cascade cipher* or *superencryption* (this term is often used in communications when some information which was encrypted off-line is transmited via an encrypted communication link). Multiple encryption is often used to enhance security of a single encryption function, though at the expense of speed. Note that multiple encryption does not necessarily improve security; consider for example

simple substitution ciphers, where the product of substitutions is another substitution. One may prove that multiple encryption is at least as secure as the first component [1, 2] if the keys are chosen independently. A typical example of multiple encryption is Triple-DES.

Alex Biryukov

References

[1] Even, S. and O. Goldreich (1985). "On the power of cascade ciphers." *ACM Transactions on Computer Systems*, 3, 108–116.

[2] Maurer, U.M. and J.L. Massey (1993). "Cascade ciphers: The importance of being first." *Journal of Cryptology*, 6 (1), 55–61.

[3] Shannon, C.E. (1949). Communication theory of secrecy systems." *Bell System Technical Journal*, 28, 656–715.

PROOF OF KNOWLEDGE VS. PROOF OF MEMBERSHIP

A basic interactive proof for a statement of the form $x \in L$, where L is some formal language is often called a *proof of membership*. For instance, if L_H is the language of graphs containing a Hamiltonian cycle, a proof for the statement $x \in L_H$ shows that x is a Hamiltonian graph. However, the proof does not necessarily show that the prover actually *knows* a Hamiltonian cycle for the graph.

A *proof of knowledge* is similar to an interactive proof, except that the soundness condition is strengthened in the following way. Referring to the example statement $x \in L_H$, the verifier accepts the proof only if the prover actually knows a Hamiltonian cycle for x, where "knowing" is characterized as follows. Generally speaking, if one is given access to a successful prover for the statement $x \in L_H$, then one should be able to extract a Hamiltonian cycle for x. Such an efficient algorithm for extracting a Hamiltonian cycle (in this example) from a prover is in general called a *knowledge extractor*.

The formal notion of a proof of knowledge developed from the results in [1–5]. Many zero-knowledge proofs of membership are in fact zero-knowledge proofs of knowledge. The Fiat–Shamir identification protocol is an early example of a zero-knowledge proof of knowledge. The Schnorr identification protocol is another well-known example.

Berry Schoenmakers

References

[1] Bellare, M. and O. Goldreich (1992). "On defining proofs of knowledge." *Advances in Cryptology—CRYPTO'92*, Lecture Notes in Computer Science, vol. 740 ed. E.F. Brickell. Springer-Verlag, Berlin, 390–420.

[2] Feige, U., A. Fiat, and A. Shamir (1988). "Zero-knowledge proofs of identity." *Journal of Cryptology*, 1 (2), 77–94.

[3] Feige, U. and A. Shamir (1990). "Witness indistinguishable and witness hiding protocols." *Proceedings of 22nd Symposium on Theory of Computing (STOC'90)*. ACM Press, New York, 416–426.

[4] Goldreich, O. (2001). *Foundations of Cryptography—Basic Tools*. Cambridge University Press, Cambridge.

[5] Goldwasser, S., S. Micali, and C. Rackoff (1989). "The knowledge complexity of interactive proof systems." *SIAM Journal on Computing*, 18, 186–208. Preliminary version in *17th ACM Symposium on the Theory of Computing*, 1982.

PROPAGATION CHARACTERISTICS OF BOOLEAN FUNCTIONS

Let n and l be positive integers such that $l \leq n$. Let f be a Boolean function on F_2^n and let $a \in F_2^n$. We denote by $D_a f$ the *derivative* of f with respect to a, that is: $D_a f(x) = f(x) \oplus f(a + x)$. This notion is related to the differential attack (see Differential cryptanalysis for block ciphers).

The function f satisfies the *propagation criterion PC(l)* of degree l if, for all $a \in E$ of weight at most l, the function $D_a f$ is balanced (that is, takes the values 0 and 1 equally often) [3]. In other words, f satisfies *PC(l)* if the autocorrelation coefficient $\sum_{x \in F_2^n} (-1)^{f(x) \oplus f(x+a)}$ is null for every $a \in F_2^n$ such that $1 \leq w_H(a) \leq l$. The *strict avalanche criterion SAC* corresponds to *PC*(1).

The functions satisfying *PC(n)* are the bent functions (see Nonlinearity of Boolean functions).

If n is even, then $PC(n-2)$ implies $PC(n)$. Since bent functions are never balanced, there exist balanced n-variable $PC(l)$ functions for n even only if $l \leq n-3$. For odd $n \geq 3$, the functions satisfying $PC(n-1)$ are those functions of the form $g(x_1 \oplus x_n, \ldots, x_{n-1} \oplus x_n) \oplus \ell(x)$, where g is bent and ℓ is affine.

The only known upper bound on the *algebraic degree* (see Boolean functions) of a $PC(l)$ function is $n-1$. A lower bound on the *nonlinearity* (see Boolean functions) of $PC(l)$ functions exists: it is lower bounded by $2^{n-1} - 2^{n-\frac{1}{2}l-1}$. Equality can occur only if $l = n-1$ (n odd) and $l = n$ (n even).

Constructions of $PC(l)$ functions have been given in [2], for instance.

It is needed, for some cryptographic applications, to have Boolean functions that still satisfy $PC(l)$ when we keep constant a certain number k of their coordinates (whatever are these coordinates and whatever are the constant values chosen for them). We say that such functions satisfy the *propagation criterion* $PC(l)$ *of order* k. This notion, introduced in [3], is a generalization of the strict avalanche criterion of order k, $SAC(k)$, introduced in [1]. $SAC(k)$ functions, that is, $PC(1)$ of order k functions, have algebraic degrees at most $n-k-1$ (see [3]). A construction of $PC(l)$ of order k functions based on Maiorana-McFarland's method is given in [2].

There exists another notion, similar to $PC(l)$ of order k, but stronger [3]: a Boolean function satisfies $EPC(l)$ of order k if every derivative $D_a f$ corresponding to a direction $a \neq 0$ of weight at most l is k-resilient.

Claude Carlet

References

[1] Forré, R. (1989). "The strict avalanche criterion: spectral properties of Boolean functions and an extended definition." *Advances in Cryptology—CRYPTO'88,* Lecture Notes in Computer Science, vol. 403, ed. S. Goldwasser. Springer-Verlag, Berlin, 450–468.

[2] Kurosawa, K. and T. Satoh (1997). "Design of $SAC/PC(\ell)$ of order k Boolean functions and three other cryptographic criteria." *Advances in Cryptology—EUROCRYPT'97*, Lecture Notes in Computer Science, vol. 1233, ed. W. Fumy. Springer-Verlag, Berlin, 434–449.

[3] Preneel, B., W. Van Leekwijck, L. Van Linden, R. Govaerts, and J. Vandevalle (1991). "Propagation characteristics of Boolean functions." *Advances in Cryptology—EUROCRYPT'90,* Lecture Notes in Computer Sciences, vol. 473, ed. I.B. Damgård. Springer-Verlag, Berlin, 161–173.

PROTOCOL

In the context of cryptography, "protocol" is a shorthand for "cryptographic protocol". A cryptographic protocol is a distributed algorithm describing precisely the interactions of two or more entities to achieve certain security objectives. The entities interact with each other by exchanging messages over private and/or public communication channels.

Important classes of protocols are: key exchange protocols or key establishment protocols, challenge–response protocols, identification verification protocols, and zero-knowledge protocols. A more practical example is the Secure Socket Layer (SSL) protocol for establishing a secure communication link between two entities, a client and a server.

Cryptographic protocols are often used as building blocks for constructing cryptographic schemes (or systems). In general, a cryptographic scheme may be composed of several cryptographic algorithms and/or cryptographic protocols.

Berry Schoenmakers

Reference

[1] Menezes, A.J., P.C. van Oorschot, and S.A. Vanstone (1997). *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL.

PROTON

Proton is a proprietary e-cash smart card solution developed by Banksys in Belgium, which is CEPS and EMV compliant. The security is based on symmetric cryptography, which basically means that the security relies on the protection of one key which has been used to encrypt data on the cards.

The card may be uploaded at an ATM or similar device. The cardholder keys in his *PIN* (see Personal Identification Number) and selects the account from which the amount should be debited.

The ATM or load device then links with the smart card chip and loads the amount onto the chip. The card can be used at stores, payphones, vending machines, pay-per-view TV set-top box, and Internet low-value payments, and the solution has been employed in a number of countries.

Transactions are fully accountable, with centrally available audit trail, but only small amounts are allowed, and payments are provided without the use of PIN-codes for cost limitation and user friendliness.

Peter Landrock

PSEUDO-MERSENNE PRIME

A prime of the form

$$p = 2^m - k,$$

where k is an integer for which

$$0 < |k| < 2^{\lfloor m/2 \rfloor}.$$

If $k = 1$, then p is a Mersenne prime (and m must necessarily be a prime). If $k = -1$, then p is called a *Fermat prime* (and m must necessarily be a power of two).

Pseudo-Mersenne primes are useful in public-key cryptography because they admit fast modular reduction (see modular arithmetic) similar to Mer- senne primes. If n is a positive integer less than p^2, then n can be written

$$n = u \cdot 2^{2m} + a \cdot 2^m + b,$$

where $u = 0$ or 1 and a and b are nonnegative integers less than 2^m. (It is only rarely true that $u = 1$, and never true if $k > 0$.) Then

$$n \equiv u \cdot k^2 + a \cdot k + b \pmod{p}.$$

Repeating this substitution a few times will yield n modulo p. This method of modular reduction requires a small number of additions and subtractions rather than the usual integer division step.

This method works best when m is a multiple of the word size of the machine being used and k is expressible in one machine word. Thus the first recorded use of pseudo-Mersenne primes in elliptic curve cryptography [1] specifies that $|k| < 2^{32}$.

More recently, pseudo-Mersenne primes have been used in the construction of optimal extension fields. These are fields of the form $\mathbb{F}_p[\alpha]$ where p is a pseudo-Mersenne prime of single word size (e.g., $2^{32} - 5$) and α is a root of an irreducible binomial equation $x^n - b$ over $\mathbb{F}_p$. Optimal extension fields are efficient fields for elliptic curve cryptography because the algebraic operations can be carried out using single-precision arithmetic modulo a pseudo-Mersenne prime (see [2]).

Jerome Solinas

References

[1] Crandall, Richard E. (1992). "Method and apparatus for public key exchange in a cryptographic system." U.S. Patent # 5,159,632, October 27, 1992.

[2] Bailey, Daniel and Christof Paar (1998). "Optimal extension fields for fast arithmetic in public-key algorithms." *Advances in Cryptology—CRYPTO'98*, Lecture Notes in Computer Science, vol. 1462, ed. H. Krawczyk. Springer-Verlag, Berlin, 472–485.

PSEUDO-NOISE SEQUENCE (PN-SEQUENCE)

Sequences which obey the three randomness postulates of Golomb are usually called pseudo-noise sequences (PN-sequences). The primary examples of such sequences are the maximum-length linear sequences, i.e., the m-sequences (see also Golomb's randomness postulates).

Tor Helleseth

PSEUDONYMS

In the paper based world, a pseudonym is a ficticious name for an individual. Examples are pen names, aliases, legalized names, or working names. In the realm of electronic communication, a pseudonym (or sometimes just 'nym') can be any identifier that an individual or object uses in a particular context. The purpose of using pseudonym is to identify an individual or object in the respective context while not identifying it in other (unintended) contexts. In effect, one keeps transactions in different contexts separate such that observers who have access to both contexts cannot link the transactions of the same individual or those related to the same object (see anonymity and unlinkability).

In oppressive political environments, people use pseudonyms to make public political statements and to hide the identities of their correspondents. In more tolerant environments people use pseudonyms to avoid embarrassment, harassment, or even loss of their jobs. Examples are discussing alcoholism, depression, sexual preferences, etc. Several servers for managing e-mail pseudonyms are available on the internet [4]. Pseudonyms are a dual use technology with many very beneficial uses and a lot of serious misuses. See Froomkin [3].

The following types of pseudonyms can be distinguished:

Person pseudonym: A pseudonym that is used over an individual's life-time or remaining life-time. Examples are author names, working names, social security numbers, driver's licence numbers, photo ID (to a certain extent), fingerprint, genetic print, etc.

Member (role) pseudonym: A pseudonym associated to an individual for as long as the individual acts as a member, employee, volunteer, or under a contract. Examples are bank account numbers, credit card numbers, e-mail

addresses, *PGP* public keys (see Pretty Good Privacy), customer numbers, cell phone numbers, member IDs for affiliations in universities, colleges, or other educational programs, membership in sports clubs, etc.

Relationship pseudonym: A pseudonym used by one person A in all communications with another person B. If B is a bank and an insurance and A is a customer of B and holds an insurance police with B then A might want to use the same relationship pseudonym with A. For example, because B can be expected to find out anyway that the account holder and the insurance police holder are the same person.

Session pseudonym: Usually a more technical kind of pseudonym because it is used only for the duration of a communication session. For example, a session ID in a TCP/IP protocol stack that is kept for the duration of one session.

Transaction pseudonym: A technical kind of pseudonym that is used only for a single transaction. For example, electronic coins can be withdrawn for one transaction pseudonym and be spent for another transaction pseudonym.

According to Pfitzmann and Köhntopp [5], these types of pseudonyms can be related in order of the increasing anonymity and unlinkability they can achieve (see Figure 1). Pseudonyms are a helpful—but usually not sufficient—aid to achieve privacy including anonymity and unlinkability. They were introduced as a technical concept in cryptography by David Chaum [1,2]. Privacy is about protecting identities of indivduals against third party interests. An individual acts anonymously if it leaves no information that traces back to the individual's identity, i.e., the registered name, address, photo ID, e-mail address, etc. One way to act anonymously is to use a pseudonym that is chosen independently of the individual's identity, preferably a randomly chosen number. More privacy can be achieved by using different pseudonyms in different contexts, thereby achieving unlinkability between transactions in different contexts. For example, a patient could use a medical ID and a health insurance ID that have no commonalities, like an encoded birth date. Then if a treating physician and an employee of the health insurance collaborate without both having "more identifying material" about the patient, they could hardly figure out that they are actually talking about the same patient. As Rao and Rohatgi [6] have emphasized, however, the use of pseudonyms alone is hardly ever enough to achieve effective anonymity or unlinkability, because in real life transactions carry a lot of context information that can be analyzed and linked to the context information of other transactions.

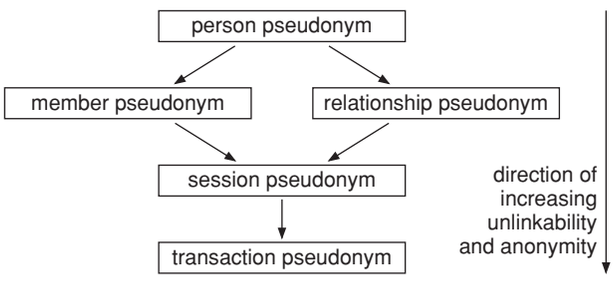

Fig. 1. Types of pseudonyms

In cryptography, pseudonyms have important applications in untraceable electronic cash, unlinkable credentials, electronic voting schemes, secure auction systems, and blind signatures.

Gerrit Bleumer

References

[1] Chaum, David (1981). "Untraceable electronic mail, return addresses, and digital pseudonyms." *Communications of the ACM*, 24 (2), 84–88.

[2] Chaum, David (1986). "Showing credentials without identification—signatures transferred between unconditionally unlinkable pseudonyms." *Advances in Cryptology—EUROCRYPT'85*, Lecture Notes in Computer Science, vol. 219, ed. F. Pichler. Springer-Verlag, Berlin, 241–244.

[3] Froomkin, Michael (1996). "Flood control on the information ocean: Living with anonymity, digital cash, and distributed databases." *University of Pittsburgh Journal of Law and Commerce*, 395 (15).

[4] Mazieres, David, Frans Kaashoek (1998). "The desgin, implementation and operation of an email pseudonym server." *5th ACM Conference on Computer and Communications Security*. ACM Press, New York, 27–36.

[5] Pfitzmann, Andreas, Marit Köhntopp (2001). "Anonymity, unobservability, and pseudonymity—a proposal for terminology." *Designing Privacy Enhancing Technologies*, Lecture in Computer Science, vol. 2009, ed. H. Frederrath. Springer-Verlag, Berlin, 1–9.

[6] Rao, Josyula R., Pankaj Rohatgi (2000). "Can pseudonyms really guarantee privacy?" *9th Usenix Symposium*, August.

PSEUDOPRIME

A *pseudoprime* (without any other qualification) is a *composite* that passes the *Fermat primality test*. More precisely, n is a *pseudoprime*

to the base a (noted as a-PSP) if n is *composite* and $a^{n-1} \equiv \mod n$ (see modular arithmetic). Similarly, a *strong pseudoprime* (SPSP) is a composite that passes the Miller–Rabin probabilistic primality test (also known as the *strong primality test*). Other qualifications for the term *pseudoprime* exist for composites that are declared to be prime by other probabilistic primality tests. The term *probable prime* is related but is not restricted to composite numbers.

Anton Stiglic

PSEUDORANDOM FUNCTION

A *pseudorandom function* is a deterministic function of a key and an input that is indistinguishable from a truly random function of the input. More precisely, let s be a security parameter, let K be a key of length s bits, and let $f(K, x)$ be a function on keys K and inputs x. Then f is a pseuodorandom function if:

- f can be computed in polynomial time in s; and
- if K is random, then f cannot be distinguished from a random function in polynomial time.

In this context, "distinguishability" refers to the ability of an algorithm to tell whether a function is not truly random. Let g be a truly random function of x with the same output length as f. Suppose a polynomial-time algorithm A is given access to a "oracle" which, on input x, either consistently returns $f(K, x)$, or consistently returns $g(x)$. After some (polynomial) number of accesses to the oracle, the algorithm outputs a guess, b, as to whether the oracle is f or g. Let ϵ be A's *advantage*, i.e., the difference in probabilities

$$\epsilon = \left|\Pr[b = \text{“}f\text{”} \mid \text{oracle is } f] - \Pr[b = \text{“}f\text{”} \mid \text{oracle is } g]\right|.$$

If the inverse $1/\epsilon$ grows faster than any polynomial in s for all polynomial-time algorithms A, then the function f is said to be indistinguishable from a random function.

Pseudorandomness is a stronger requirement than being a *one-way function*, since a one-way function only needs to be hard to invert; a pseudorandom function also needs to be hard to *guess* when the key is unknown.

Pseudorandom functions have many applications in cryptography, as they provide a way to turn an input into a value that is effectively random. This is helpful for computing MAC algorithms, deriving keys from other keys, and more generally for replacing random number generators in an application with a deterministic function, when a secret key is available. Goldreich, Goldwasser and Micali [1] showed how to construct a pseudorandom function from a pseudorandom number generator; in practice, a pseudorandom function is often constructed from a hash function, as, for example, in the popular Secure Sockets Layer (SSL) protocol.

Burt Kaliski

Reference

[1] Goldreich, O., S. Goldwasser, and S. Micali (1986). "How to construct random functions." *Journal of the ACM*, 33 (4), 210–217.

PSEUDO-RANDOM NUMBER GENERATOR

Many cryptographic primitives require random numbers, to be used as keys, challenges, unique identifiers, etc. However, generating random values on a computer is in fact a very difficult task. One possible method for such generation is to use a pseudo-random numbers generator.

A pseudo-random number generator (PRNG) is a function that, once initialized with some random value (called the *seed*), outputs a sequence that appears random, in the sense that an observer who does not know the value of the seed cannot distinguish the output from that of a (true) random bit generator.

It is important to note that a PRNG is a deterministic process: put back in the same state, it will reproduce the same sequence, as will two PRNGs initialized with the same seed. This property makes PRNGs suitable for use as stream ciphers.

Pseudo-random generators aimed for cryptographic applications must not be confused with those used for most other purposes (simulations, probabilistic algorithms, ...). Whereas the latter must basically have good properties in terms of the statistical distribution of their output, the former must also resist against active adversaries disposing of an extended description of the generator's structure, observing a large quantity of output, and using advanced cryptanalysis methods in order to predict future output. Good statistical distribution is a necessary, but not sufficient condition for this. Classical PRNGs do not resist such

attacks, and specific (namely cryptographically secure) PRNGs must be used for security purposes.

An important notion related to random numbers generation is that of *entropy* (see information theory), that basically measures the amount of information carried by a sequence (or, in other words, the amount of *a priori* uncertainty about a sequence). Since the sequence generated by a PRNG is entirely determined by its seed, the entropy cannot be greater than that of the seed, no matter how long the generated sequence is. This distinguishes PRNGs from a (true) random bit generator. However, provided that the seed is long enough to make exhaustive search beyond the reach of today's computing power, this limitation does not rule out the use of PRNGs in some cryptographic applications.

PRNG STRUCTURE: A PRNG is typically composed of a seed repository, an output function producing some random-looking bits from the seed, and a feedback function that iteratively transforms the seed.

It can be shown that a PRNG is necessarily periodic: the sequence it produces will repeat itself after a (possibly extremely long) period. Care must be taken that the generator is not used to produce more data than a full cycle.

PRNG EXAMPLES: This section reviews some well-known PRNGs constructs, although the list is in no way deemed to be exhaustive.

Linear Feedback Shift Registers

A very simple and efficient (but insecure) construct is given by linear feedback shift registers (LFSRs). A LFSR is composed of a register, which is an array of memory cells, each capable of storing one binary value, and a feedback function, which consists in the XOR operator applied to selected cells of the register (Figure 1). For each new unit of time, the following operations are performed:

1. The content of the last memory cell is output.
2. The register is processed through the feedback function. In other words, selected memory cells are XORed together to produce one bit of feedback.
3. Each element of the register advances one position, the last element being discarded, and the first one receiving the result of the feedback function.

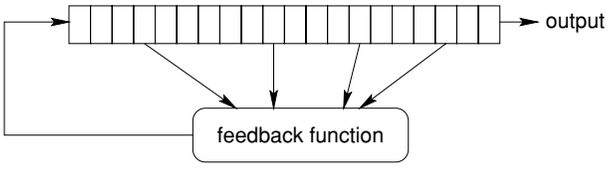

Fig. 1. Linear feedback shift register

LFSRs can be implemented very efficiently in hardware, can produce sequences with large periods and good statistical distribution, and are theoretically well understood. However, they are not cryptographically secure: efficient techniques are known to reconstruct the content of the register, and hence the full LFSR's output, based on the observation of a short output sequence.

Nevertheless, LFSRs can be used as building blocks to construct a secure PRNG. Common combination techniques consist in:

- using a nonlinear function to combine the output of several LFSRs;
- using the output of one (or a combination of) LFSR(s) to clock one (or a combination of) other LFSR(s).

However, great care has to be taken in the design of such a combination, which must be carefully analyzed against known cryptanalytic techniques. Many designs prove much less secure than they first appear, as is for example witnessed by the Geffe generator [2], that succumbs to correlation attacks [5].

Blum–Blum–Shub Generator

The Blum–Blum–Shub PRNG is a strong generator, in the sense that it has been proved secure under the assumption that integer factorization is intractable [8]. However, its slowness limits its usefulness to very specific cases.

The generator works by repeatedly squaring a (secretly-initialized) value modulo n (see modular arithmetic), and outputting its least significant bit (Algorithm 1). A modular squaring operation is thus required for each output bit. More efficient versions, outputting more than one bit per squaring, have also been proved secure [5].

ALGORITHM 1. Blum–Blum–Shub pseudo-random generator

Generate two large secret primes p and q, such that $p \equiv 3 \bmod 4$ and $q \equiv 3 \bmod 4$.
Compute $n = pq$
Generate a random integer s such that $\gcd(s,n) = 1$
$x_0 \leftarrow s^2 \bmod n$
for $i \leftarrow 1$ to l **do**
  $x_i \leftarrow x_{i-1}^2 \bmod n$
  Output the least significant bit of x_i
end for

Other Constructions

Another frequently used structure for PRNGs consists in processing a register through a one-way function, using (all or part off) the result as output. The register is then updated, either using a counter method (e.g., increase register by one), or by copying part of the one-way function's output into it.

This is for example the case of the ANSI X9.17 generator (based on the Data Encryption Standard, or of the FIPS 186 generator (based on the Data Encryption Standard or the hash function SHA). A detailed description of these algorithms can be found in [5].

STATISTICAL TESTS: Although it does not constitute a sufficient condition for a PRNG to be cryptographically secure, passing statistical tests is certainly necessary, since any statistical defect can be used by an attacker to gain information about the PRNG's future output. Statistical tests thus provide a first indication on the quality of a generator, although it has to be completed by a careful analysis of the generator's structure and its resistance to cryptanalysis.

Several test suites have been proposed. Among the most well-known, we can cite:

- Golomb's randomness postulates were one of the first attempts to establish necessary conditions for randomness. Nowadays, they are of mere historical interest.
- **Knuth** [3] proposes a set of simple randomness tests to apply to a sequence.
- **FIPS 140-1** [6] is a standard test series defined by the U.S. National Institute of Standards and Technology (NIST), inspired by Knuth's basic tests. This standard was superseded by FIPS 140-2.
- **FIPS 140-2** [7] (see FIPS 140) defines a more comprehensive battery of tests, including Maurer's Universal Statistical test.
- **Maurer's Universal Statistical test** [4] is a test capable of detecting a large class of defects. In addition, this test provides an estimate of the entropy of the source. A better estimate can be obtained using a modified version of the test [1].

PRNG INITIALIZATION: Since the output is entirely determined by the value of the seed, it is very important for the seed to be initialized to some truly random, unpredictable value (see random bit generator). A PRNG can be viewed as a tool expanding a small random value into a much longer sequence.

This random initialization can be performed only once (e.g. during on-factory personalization phase). Other applications require occasional PRNG re-seeding.

Generating this truly random seed is usually outside the scope of PRNG definitions that rarely address the issue. However, its importance must not be underestimated.

François Koeune

References

[1] Coron, J.-S. (1999). "On the security of random sources." *Proceedings of PKC'99*, Lecture Notes in Computer Science, vol. 1560, eds. H. Imai and Y. Zheng. Springer-Verlag, Berlin.

[2] Geffe, P. (1973). "How to protect data with ciphers that are really hard to break." *Electronics*, 46, 99–101.

[3] Knuth, D.E. (1997). *The Art of Computer Programming* (3rd. ed.). *Computer Science and Information Processing*, *Vol. 2. Seminumerical Algorithms*. Addison-Wesley, Reading, MA.

[4] Maurer, U.M. (1992). "A universal statistical test for random bit generators." *Journal of Cryptology*, 5 (2), 89–105.

[5] Menezes, A.J., P.C. van Oorschot, and S.A. Vanstone, (1997). *Handbook of Applied Cryptography*. CRC Press, Boca Raton, FL.

[6] US Department of Commerce—National Institute of Standards and Technology (1994). *FIPS 140-1: Security requirements for Cryptographic Modules*, available at http://csrc.nist.gov/publications/fips/fips1401.pdf

[7] US Department of Commerce (1999). *FIPS 140-2: Security requirements for Cryptographic Modules (Draft)*, available at http://csrc.nist.gov/publications/fips/dfips140-2.pdf

[8] Vazirani, U.V. and V.V. Vazirani (1985). "Efficient and secure pseudo-random number generation." *Advances in Cryptology—CRYPTO'84*, Lecture Notes in Computer Science, vol. 196, eds. G.R. Blakley and D. Chaum. Springer, Berlin, 193–202.

PUBLIC KEY CRYPTOGRAPHY

Public key cryptography is a method to encrypt messages using a nonsecret key. The term public key cryptography also includes various others cryptographic methods using a nonsecret key, such as authentication, digital signature schemes, and key agreement. Here we describe public key encryption schemes.

Conventional cryptography, also known as symmetric cryptosystem, uses a secret key to encrypt messages; the same key is required to decrypt these messages. In a public key encryption scheme however, knowledge of the key used to encrypt messages (which we call *encryption key* in the sequel) does not allow one to derive the key to decrypt the messages. Therefore an encryption key can be made public without endangering the security of the decryption key. A pair of encryption key and decryption key is generated for each receiver, and all the encryption keys are published. When sending a secret message to a receiver, the sender picks the public encryption key of the receiver and encrypts the message with it. The encrypted message can be recovered by using the corresponding secret decryption key, which only the intended receiver has.

Public key cryptography solves the key agreement problem and key management problem of conventional, symmetric, key cryptosystems, where it is the concern of a sender to agree with the receiver on a secret encryption key. Moreover, they both have to store these keys secretly while the number of the keys grows linearly in the number of possible receivers. The disadvantage of public key cryptography is in its processing speed. Therefore it is often used to for key agreement purposes only. A secret key that is used to encrypt a message via symmetric key cryptography is encrypted using public key cryptography and is attached to the encrypted message. The receiver first decrypts the secret key using his own decryption key and then decrypts the message using the recovered secret key.

The design of a public key cryptosystem can be based on a trapdoor one-way function. Some examples of public key encryption schemes are: RSA public key encryption, ElGamal public key encryption, Rabin cryptosystem, Goldwasser–Micali encryption scheme, Blum–Goldwasser public-key encryption scheme, and Paillier encryption and signature schemes.

A public key encryption scheme is comprised of three algorithms: a *key generation algorithm*, an *encryption algorithm* and a *decryption algorithm*. The security of a public key encryption scheme is evaluated through security measures called semantic security. If an encryption scheme is semantically secure, then an adversary, who is given an encryption of either one of the two plaintexts, cannot guess which one is encrypted with a probability more than 1/2.

In practice, it is important that the published public key is indeed the key of the intended receiver, i.e., that the key is certified (see certificate). Otherwise, a sender could be encrypting the classified message using the wrong key, and the message could be obtained by the adversary. In order to certify keys, the notion of a public key infrastructure has been developed.

Kazue Sako

PUBLIC KEY INFRASTRUCTURE

This term, or PKI, is used as a label for any security architecture where end-users have digital signature capability and possibly encryption as well based on public key cryptography. It is a general misconception that such solutions always require the use of certificates. Certificates are only required if end-users have to communicate with more than a few entities, e.g., with each other. Examples of PKI solutions without certificates are electronic banking solutions where customers may generate signatures but only communicate with the bank. But is possible to build complex PKIs as well without certificates, where instead an online service can provide the status of a public key at any point in time (so-called instant certificates). In more conventional solutions, the infrastructure is based on some general standard, such as X.509 or EMV, where each user has a certificate on each of his public key(s). If certificates are required, it is often necessary to provide means for verifying whether a certificate—which on the face of it appears to be valid—has not been revoked for some reason. This is handled by means of revocation lists or on-line inquiry protocols regarding the status of a certificate (such as *OCSP*, the On-line Certificate Status Protocol).

Peter Landrock

PUBLIC KEY PROXY ENCRYPTION

In a public key proxy encryption scheme as introduced by Blaze and Strauss [1, 2] there are a collection of recipients who generate their public key pairs of public encryption keys and private decryption keys (see public key cryptography) and one or more proxy principals who are provided with proxy keys. A proxy key $\pi_{A\to B}$ with respect to recipients A and B allows a proxy principal to convert

ciphertext that can be decrypted by A into ciphertext that can be decrypted by B giving the same plain text. The conversion of ciphertexts works such that the proxy principal learns no information about the underlying plaintext, which implies that the proxy principal does not learn enough information about the deciphering keys of A and B.

Note that proxy encryption is different from passing the decryption key of A and the encryption key of B to the proxy principal, such that the proxy principal first decrypts an incoming message using the decryption key of A and then re-encrypts the plain text to B. In this case, the proxy principal would learn all plain texts underlying the cipher texts being transformed. In addition, proxy encryption is usually a faster operation than first decrypting and afterwards re-encrypting again.

Compared to a conventional public key encryption scheme, the confidentiality requirement of a public key proxy encryption scheme holds not only against eavesdroppers and unintended recipients, but also against distrusted proxy principals. The main security requirement on public key proxy encryption schemes is:

Confidentiality: An attacker who has access to public encryption keys, proxy keys, and cipher texts, cannot figure information about the corresponding plain texts, which implies that he cannot recover information about the corresponding decryption keys.

Historically, the term *proxy cryptosystem* was introduced by Mambo and Okamoto [4], but their trust model is the same as that of conventional public key encryption. They only consider senders and recipients, but not separate proxy principals that are distrusted by the recipients $A, B, \ldots$.

Public key proxy encryption schemes have been developed by Blaze and Strauss [1, 2] into a separate class of cryptographic schemes. They have categorized public key proxy encryption schemes as follows.

Trust model between recipients: Clearly, if Alice empowers a proxy to divert her cipher texts to another recipient Bob, then she trusts Bob. But Bob does not need to trust Alice. More precisely a public key proxy encryption scheme is called *symmetric* if Alice can figure Bob's private decryption key if she learns the proxy key $\pi_{A\to B}$. Otherwise, it is called *asymmetric*.

Input to proxy key computation: An asymmetric public key proxy encryption scheme is called *active* if it is feasible to compute a proxy key $\pi_{A\to B}$ only if Bob provides his private decryption key as an input. If it is feasible to compute a proxy key $\pi_{A\to B}$ without the input of Bob's private decryption key (but only his public encryption key), then the scheme is called *passive*.

Anonymity of proxy keys: Public key proxy encryption schemes can be categorized according to how much information they reveal about the public encryption keys of the two recipients who introduced the proxy key. A public key proxy encryption scheme is called *transparent* if a proxy key $\pi_{A\to B}$ reveals the public keys of Alice and Bob. A scheme is called *translucent* if an attacker who guesses the public encryption keys of Alice and Bob can verify his guess by using the proxy key $\pi_{A\to B}$. A scheme is *opaque*, if a proxy key $\pi_{A\to B}$ reveals no information about the public encryption keys of Alice and Bob.

To illustrate the concept of proxy encryption, we consider a simple proxy encryption scheme proposed by Blaze and Strauss, which is based on a variant of the ElGamal encryption scheme. Consider p a large safe prime, q a large prime factor of $p-1$, and g be an element of the group $\mathbb{Z}_p^*$ such that g is of order q, i.e., q is the smallest positive integer such that $g^q = 1 \pmod{p}$ (see modular arithmetic). Alice and Bob be two recipients with respective public key pairs (x_i, y_i) $(i \in \{A, B\})$, where the decryption keys x_i are chosen independently and uniformly at random from $\mathbb{Z}_q^*$ and the encryption keys are computed such that $y_i = g^{x_i} \pmod{p}$. Alice can encrypt a message m for Bob by computing the ciphertext $c = (c_1, c_2) = (mg^k, y_B^k)$ consisting of two components, and Bob can decrypt the ciphertext by computing $m' = c_1/c_2^{a^{-1}} \pmod{p}$.

In order to install a proxy between them, Alice and Bob compute the proxy key $\pi_{A\to B} = x_B/x_A \pmod{q}$. If neither Alice nor Bob trusts the other one, they could use a two party computation protocol (see multiparty computation) whose output is given to the proxy. This way, neither Alice nor Bob learns the private decryption key of the other party and the proxy learns none of the private decryption keys.

If $c_A = (mg^k, y_A^k)$ is an encrypted message m to Alice, then the proxy can transform it into an encrypted message m to Bob as follows:

$$c_B = \left(c_{A1}, c_{A2}^{\pi_{A\to B}}\right) = (mg^k, g^{x_A k x_B/x_A}) = (mg^k, y_B^k).$$

This scheme is shown by Blaze and Strauss [1] to achieve confidentiality in the above sense under the computational Diffie–Hellman assumption and the discrete logarithm problem. It is symmetric because Alice can easily figure Bob's private decryption key if she learns the proxy key $\pi_{A\to B} = x_B/x_A \pmod{q}$ because she knows her own private decryption key x_A. It is active because computing the proxy key requires Bob's private decryption key x_B as input.

No asymmetric public key proxy encryption scheme has been proposed to date.

Blaze and Strauss [1] propose proxy encryption as a useful mechanism in restricted computing environments such as smart cards. Girard proposes to use proxy encryption in the personalization process of smart cards [3]. Dually related to the notion of public key proxy encryption is that of public key proxy signatures.

Gerrit Bleumer

References

[1] Blaze, Matt and Martin Strauss (1998). "Atomic proxy cryptography." Technical Report 98.5.1, AT&T Labs Research, http://www.research.att.com/library/trs

[2] Blaze, Matt, Gerrit Bleumer, and Martin Strauss (1998). "Divertible protocols and atomic proxy cryptography." *Advances in Cryptology—EUROCRYPT'98*, Lecture Notes in Computer Science, vol. 1403, ed. K. Nyberg. Springer-Verlag, Berlin, 127–144.

[3] Girard, Pierre (2000). "Secure personalization using proxy cryptography." *CARDIS'98*, Lecture Notes in Computer Science, vol. 1820, eds. J.-J. Quisquater, B. Schneier. Springer-Verlag, Berlin, 326–335.

[4] Mambo, Masahiro and Eiji Okamoto (1997). "Proxy cryptosystem: Delegation of the power to decrypt ciphertexts." *IEICE Transaction*, E80-A (1), 54–63. http://search.ieice.org/1997/pdf/a010054.pdf

PUBLIC KEY PROXY SIGNATURES

In a public key proxy signature scheme as introduced by Blaze and Strauss [1, 2] there are a collection of signers who generate their public key pairs of public verification keys and private signing keys (see public key cryptography) and one or more proxy principals who are provided with proxy keys. A proxy key $\pi_{A\to B}$ with respect to signers A and B allows a proxy principal to convert a digital signature valid with respect to A's verifying key into a signature for the same message that is valid with respect to B's verifying key. The conversion of signatures works such that the proxy principal learns no information about the signing keys of either A or B.

Compared to a conventional public key signature scheme, the unforgeability requirement of a public key proxy signature scheme holds not only for general active attackers without access to the signers private key, but also for distrusted proxy principals. The main security requirement on public key proxy signature schemes is:

Unforgeability: An attacker who has access to public verifying keys, proxy Keys, and pairs of messages with valid signatures (or can adaptively choose the messages for which he gets valid signatures) cannot produce signatures for any new messages, which implies that he cannot gather information about the corresponding private signing keys.

Historically, the term *proxy signatures* was introduced by Mambo and Okamoto [3], but their trust model is the same as that of conventional digital signatures. They only consider signers and verifiers, but not separate proxy principals that are distrusted by the signers $A, B, \ldots$.

Proxy signature schemes have been developed by Blaze and Strauss [1, 2] into a separate class of cryptographic schemes. They have categorized proxy signature schemes as follows.

Trust model between recipients: Clearly, if Bob empowers a proxy to transform Alice's signatures into Bob's own signatures, then he clearly trusts Alice. But Alice does not need to trust Bob. More precisely a proxy signature scheme is called *symmetric* if Bob can figure out Alice's private signing key if he learns the proxy key $\pi_{A\to B}$. Otherwise, it is called *asymmetric*.

Input to proxy key computation: An asymmetric proxy signature scheme is called *active* if it is feasible to compute a proxy key $\pi_{A\to B}$ only if Alice provides her private signing key as an input. If it is feasible to compute a proxy key $\pi_{A\to B}$ without the input of Alice's private signing key (but only her public verifying key), then the scheme is called *passive*.

Anonymity of proxy keys: Proxy signature schemes can be categorized according to how much information they reveal about the public verifying keys of the two recipients who introduced the proxy key. A proxy signature scheme is called *transparent* if a proxy key $\pi_{A\to B}$ reveals the public verifying keys of Alice and Bob. A scheme is called *translucent* if an attacker who guesses the public verifying keys of Alice and Bob can verify his guess by using the proxy key $\pi_{A\to B}$. A scheme is *opaque*, if a proxy key $\pi_{A\to B}$ reveals no information about the public verifying keys of Alice and Bob.

To illustrate the concept of proxy signatures, we consider a simple proxy signature scheme, which is based on a variant of the ElGamal signature scheme. Consider p a large safe prime number, q a large prime factor of $p-1$, and g be an element of the group $\mathbb{Z}_p^*$ such that g is of order q, i.e., q is the smallest positive integer such

that $g^q = 1 \pmod{p}$. Let Alice and Bob be two signers with respective public key pairs (x_i, y_i) $(i \in \{A, B\})$, where the private signing keys x_i are chosen independently and uniformly at random from $\mathbb{Z}_q^*$ and the public verification keys are computed such that $y_i = g^{x_i} \pmod{p}$. $h(m)$ denotes a collision resistant hash function that takes any binary string $m \in \{0,1\}^+$ as input and returns a value in $\mathbb{Z}_q$. Alice can sign a message m by choosing some $k \in_R \mathbb{Z}_q$ and computing the signature $\sigma = (r, s) = (g^k, x_A r + kh(m))$ consisting of two components, and anyone with access to Alice's verifying key y_A can verify that σ is a valid signature for m by checking that $g^s = y_A^r r^{h(m)} \pmod{p}$.

In order to install a proxy between them, Alice and Bob compute the proxy key $\pi_{A\to B} = x_B - x_A \pmod{q}$. If neither Alice nor Bob trusts the other one, they could use a two party computation protocol whose output is given to the proxy. This way, neither Alice nor Bob learns the private signing key of the other party and the proxy learns none of the private signing keys.

If $\sigma_A = (g^k, x_A r + kh(m))$ is a signature by Alice for message m, then the proxy can transform it into a signature σ_B by Bob for message m as follows:

$$\begin{aligned}\sigma_B &= (\sigma_{A1}, \sigma_{A2} + \pi_{A\to B} r)\\ &= (g^k, x_A r + kh(m) + (x_B - x_A) r)\\ &= (g^k, x_B r + kh(m)).\end{aligned}$$

This scheme is symmetric because Bob can easily figure Alice's private signing key x_A if he learns the proxy key $\pi_{A\to B} = x_B - x_A \pmod{q}$ because he knows his own private signing key x_B. It is active because computing the proxy key requires Alice's private signing key x_A as input.

No asymmetric proxy signature scheme has been proposed to date.

Dually related to the notion of proxy signatures is that of public key proxy encryption.

Gerrit Bleumer

References

[1] Blaze, Matt and Martin Strauss (1998). "Atomic proxy cryptography." Technical Report 98.5.1, AT&T Labs Research, http://www.research.att.com/library/trs

[2] Blaze, Matt, Gerrit Bleumer, and Martin Strauss (1998). "Divertible protocols and atomic proxy cryptography." *Advances in Cryptology—EUROCRYPT'98*, Lecture Notes in Computer Science, vol. 1403, ed. K. Nyberg. Springer-Verlag, Berlin, 127–144.

[3] Mambo, Masahiro, Keisuke Usuda and Eiji Okamoto (1996). "Proxy signatures: Delegation of the power to sign messages." *IEICE Transactions Fundamentals*, E79-A (9), 1338–1354. http://search.ieice.org/1996/pdf/e79-a_9_1338.pdf