Hand Geometry Recognition

David Zhang, Vivek Kanhangad
Biometrics Research Centre, Department of Computing,
The Hong Kong Polytechnic University, Hung Hom,
Kowloon, Hong Kong

Synonyms

Hand geometry verification

Definition

Hand geometry recognition is a biometric approach to automatically recognize individuals based on the unique geometric features of the hand.

Background

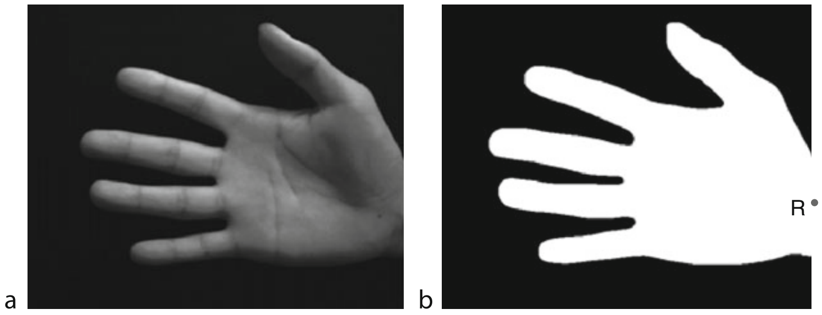
Hand geometry biometric systems utilize features such as finger length, finger width, finger thickness, finger area, and palm width to perform personal authentication. These systems have gained immense popularity and public acceptance as evident from their extensive deployment for applications in access control, time attendance applications and several other verification tasks. Major advantages of hand geometry systems include simple imaging requirements (features can be extracted from low-resolution hand images), the ability to operate under harsh environmental conditions (immune to dirt on the hand and other external factors), and low data-storage requirements. In addition, hand geometry acquisition and verification is extremely fast. These distinct advantages over other biometrics helped the hand geometry systems capture a niche market.

Hand recognition approaches can be classified in to three categories based on the nature of image acquisition:

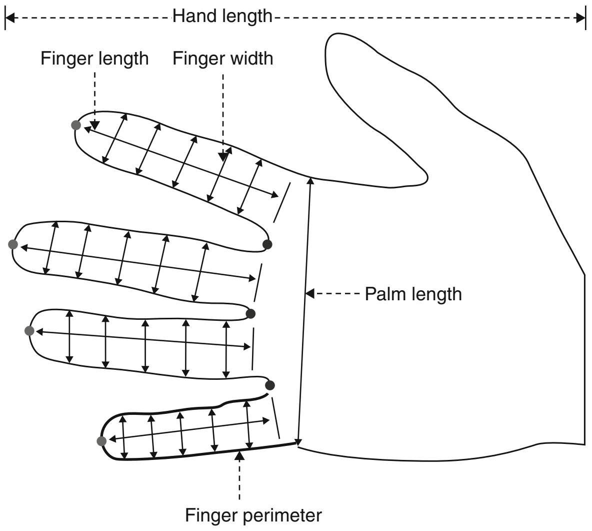
1. *Constrained and contact-based*: These systems employ pegs or pins on the platform to constrain the position and posture of the hand. Image acquisition is usually done under controlled environments, with uniform illumination. Hand images acquired using such systems require minimum processing prior to feature extraction and thereby considerably reducing the time required for the process of authentication. Typical hand geometry features extracted from the acquired hand images include finger length, finger width, finger thickness, and palm width. Finger thickness features are computed from the lateral view of the hand, which can be easily acquired using a mirror with this kind of constrained imaging setup.
2. *Unconstrained and contact-based*: Hand images are acquired in an unconstrained manner, often requiring the users to place their hand on flat surface. A back-lit surface with camera mounted on top can also be employed to acquire high-contrast hand images. Landmark points (finger tips and finger valleys) on the hand image are commonly used to align images prior to feature extraction. An alternate approach is to extract features that are invariant to image plane transformations of the acquired hand images.
3. *Unconstrained and contact-free*: This approach does away with the need for any pegs or platform during hand image acquisition. Users are given the freedom to hold their hand freely in the 3D space, usually at a fixed (approximately) distance from the camera. The absence of any mechanism to constrain the placement of the hand introduces additional challenges in terms of variations in scale and out-of-plane transformations (3D pose) in the acquired hand images. Images acquired in cluttered backgrounds may also require sophisticated segmentation algorithms to localize the hand in the image. This mode of image acquisition is believed to be more user-friendly.

Theory

A typical imaging set up for a 2D hand geometry system would involve the following components: a CCD camera, illumination source, and a flat surface [1–3]. CCD camera employed is usually low to medium resolution as the hand geometry features can be extracted from binary images of the hand. Prior to feature extraction, acquired hand images are usually processed to obtain a binary image of the hand or in some cases, a hand contour. In most cases, a simple thresholding scheme followed by morphological operations is used to segment the hand from the background (refer to Fig. 1). Boundary pixels of the hand in the processed binary image are then identified using the contour-tracing algorithm to obtain the hand contour. Figure 2 shows typical



Hand Geometry Recognition. Fig. 1 Preprocessing stages (**a**) Acquired intensity image (**b**) Binary hand image after thresholding and morphological operations



Hand Geometry Recognition. Fig. 2 Typical hand geometry features marked on a hand contour

hand geometry features extracted from the contour of the hand. Finger length is computed as the distance from the finger tip to its base along the orientation of the finger. Finger-width measurements are made at a number of evenly spaced points along the finger length. Finger perimeter refers to the number of pixels on the finger contour. All the measurements shown in Fig. 2 are usually made in terms of pixels. In addition to these geometric measurements from the hand, the hand contour (silhouette) can also be used (as a shape feature) in establishing the identity.

Experimental Results

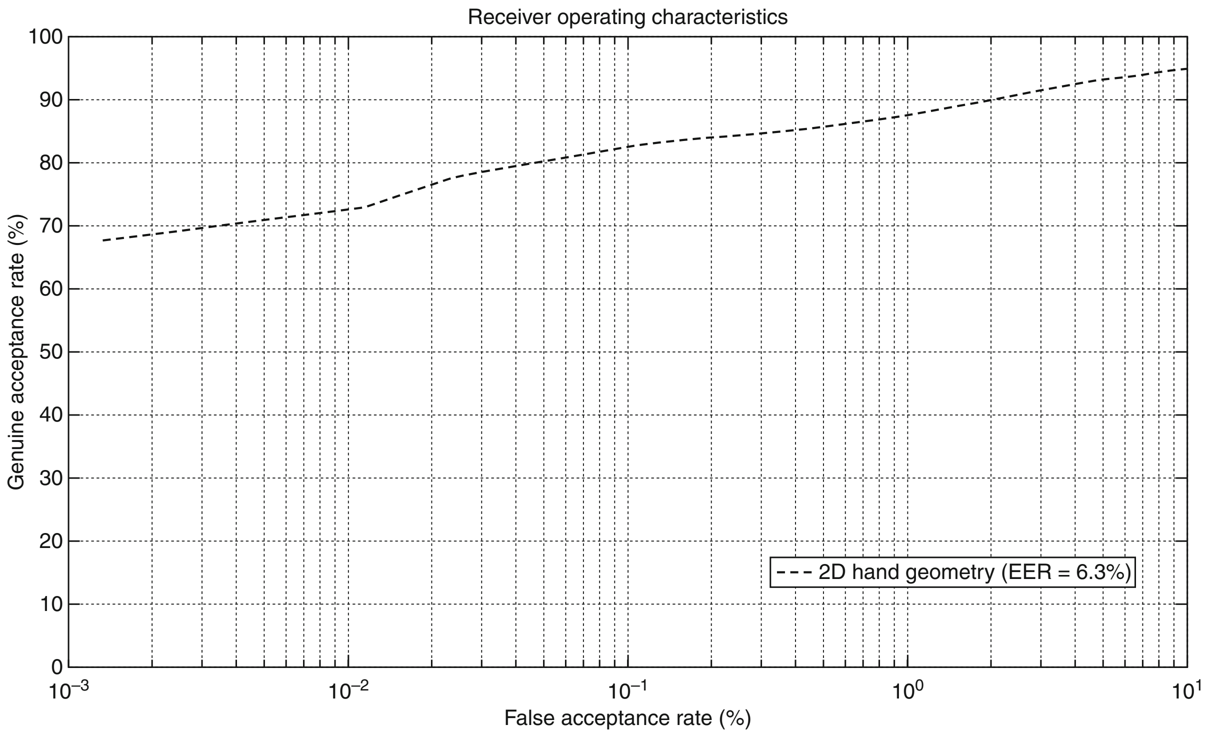
Experiments are performed on a database of hand images acquired from 177 subjects. Five samples are collected in the first session (training samples), followed by another five samples in the second session (probe samples). All of these hand images are acquired in an unconstrained and contact-free manner. The feature extraction module computed various features (refer to Fig. 2) and concatenated them to form a feature vector. The computation of matching distance between two feature vectors is based on the Euclidean distance. As shown in Fig. 3, the hand geometry matcher achieved an EER of 6.3% on the aforementioned database.

Applications

The history of hand geometry biometric technology/systems dates back over 3 decades. In fact, the hand geometry system – Identimat developed by Identimation – is one of the earliest reported implementations of a biometric system for commercial applications. Since then, the hand geometry biometric systems have found applications in wide variety of fields ranging from airports to nuclear power plants [4]. USPASS (formerly known as INSPASS) is the first and the largest hand-geometry-based biometric verification program undertaken by the US government to accelerate the process of immigration for authorized and frequent travelers. As part of this program, HandKey scanners were installed at certain airports in the USA to accelerate the process of immigration for frequent fliers. Another large-scale deployment was at 1996 Olympics Games in Atlanta, where hand geometry scanners were installed to restrict access to the Olympic Village. Due to limited discriminatory power of the hand geometry features, these systems are rarely employed for applications that require performing identity recognition from a large-scale database. However, hand geometry systems are extensively used for personal verification tasks.

Recommended Reading

1. Duta N (2009) A survey of biometric technology based on hand shape. Pattern Recogn 42(11):2797–2806
2. Jain AK, Ross A, Pankanti S (1999) A prototype hand geometry-based verification system. Proceedings of the AVBPA, Washington DC, pp. 166–171, Mar 1999



Hand Geometry Recognition. Fig. 3 Receiver operating characteristics (ROC) curve for a hand geometry matcher

3. Sanchez-Reillo R, Sanchez-Avila C, Gonzalez- Macros A (2000) Biometric identification through hand geometry measurements. IEEE Trans Pattern Anal Mach Intell 22(10):1168–1171
4. Sidlauskas DP, Tamer S (2008) Hand geometry recognition. In Jain AK, Flynn P, Ross A (eds) Handbook of Biometrics. Springer, New York

Hand Geometry Verification

►Hand Geometry Recognition

Handwriting Analysis

Patrizio Campisi[1], Emanuele Maiorana[1], Harold Mouchère[2], Christian Viard-Gaudin[2], Alessandro Neri[1]
[1] Department of Applied Electronics, Università degli Studi Roma TRE, Rome, Italy
[2] Polytech Nantes, Université de Nantes, IRCCyN, Nantes, France

Synonyms

Longhand; Penmanship

Definition

The term handwriting refers to the act of writing by hand graphical marks with the purpose of information communication according to syntactic and semantic specific rules of a specific language.

Background

The action of handwriting is controlled by a complex neuromuscular process [1] which acts on a writing tool combined with a writing surface to produce a two-dimensional trajectory. Inherently, handwriting is a continuous analog signal which can, however, be digitized in order to be processed by computers. Handwriting data acquisition can be either off-line or on-line. In the off-line case, only the completed writing is available as an image produced by a scanner or a digital camera, where the main parameter which governs the acquisition process is the sampling resolution measured in dots per inch (dpi). This image is a static representation of handwriting where all dynamic information related to handwriting are lost. On the contrary, in the on-line modality, the temporal evolution of some physical quantities involved in the act of writing are collected by means of specialized devices such as PDA, Tablet PC, digital pens, or electronic white boards during the writing process. Depending on the technology, the capturing devices can provide a variety of information such as the coordinates of the pen tip, a Boolean value indicating contact with writing plane, pressure, angles with the writing plane or tilt, distance from writing plane, etc. The basic data is a sequence of points, called stroke, which is obtained by sampling the curve at regular intervals of time. Hence, both spatial and temporal sampling rates have to be defined. Typically, 100 samples per second are collected with a spatial resolution from 100 to 300 dpi.

Theory

With the advent of electronic handwriting devices, many researchers have been attracted by the field of *automatic handwriting analysis*. The information contained in handwritten text is both *textual* information, which refers to what has been written, and *personal* information, which is related to the identity of the writer. The extraction of the textual information is addressed by *handwriting recognition*, which consists of transforming graphical marks into a symbolic representation. Therefore, the specific handwriting style, unique to each writer, is eliminated by means of some preprocessing in order to focus only on the message conveyed by the written text. Size normalization, based on the automatic extraction of the baselines or de-slanting, are some of these preprocessing techniques, which are the cornerstones of most of the handwriting recognition systems. On the other hand, when we do not care about the message itself but we aim at recovering the identity of the writer, we talk about *writer recognition*. In this latter case, it is of paramount importance to keep all distinctive features related to what has been produced by a writer hand.

Hence, handwriting recognition and writer recognition represent two opposing facets of handwriting.

Handwriting recognition has driven a lot of attention in the last decades because of practical applications such as reading handwritten texts on postal envelopes (mail sorting applications) or bank checks (legal and numerical amounts recognition), which are typical applications in the off-line domain, or input methods in the on-line domain as a substitute to keyboard entry mainly for mobile applications.

On the other hand, a growing interest has raised for the use of handwriting for writer recognition. In fact, handwriting can be considered as a biometric signal which conveys a lot of information related to the writer who has produced the questioned piece of text. More specifically, handwriting can be seen as a behavioral biometrics since it depends on many factors, like writing position, place or circumstances, health conditions, fatigue, emotional state, etc. Although the nature versus nurture properties of handwriting are questionable, all the studies that have been conducted demonstrate the high discriminative power of handwriting. At school, we all learn to write according to a standard writing style, the copy book, that varies according to the geographical location, temporal circumstances, and the cultural and historical backgrounds. With the passage of time, we develop individual writing characteristics and our handwriting starts to deviate from the learned style. These unique characteristics serve to distinguish an individual's writing from another's, even if the two writings share the same copy book, making it possible to identify the author for which one has already seen a written text. Automatic writer recognition serves as a valuable solution for the document examiners, paleographers, graphologists, and forensic experts. More specifically, it can be used for either *writer identification*, in order to determine the author of a text from a set of writers, or for *writer verification*, to decide whether or not two distinct texts have been written by the same subject.

Handwriting recognition has matured over the past few decades to a stage where readily available commercial and industrial text recognition engines are able to provide us reasonably high recognition accuracies. See for example [2–5] and references therein.

On the same trend, much progress has been made in the field of writer recognition in the last decade. Pioneering works in establishing a formal quantitative analysis of handwriting individuality can be credited to the studies in [6, 7], where pattern recognition techniques have been used to provide objective assessments and scientific validations of the individuality in handwriting. A nearest neighbor algorithm and a 3-layered neural network have been used to train and classify the results for writer identification and writer verification, respectively, using a database of 1,500 individuals stratified across different genders, age, groups, and ethnicity. The obtained results have verified the hypothesis that individuality exists in handwriting with a 95% confidence from a statistical inference of the entire US population. Writer recognition can be either text independent, when no assumptions about the underlying textual data are made, or text dependent, when the same text has to be written in the enrollment and recognition stage. The choice between text-independent or text-dependent approaches typically depends on the target application and on the available data. Signatures are examples of text-dependent systems since the writers have to write the same text they have written previously during the enrollment process. On the other hand, text-independent techniques do not bind the writers to any specific line of text in order for the system to recognize them. Instead, the system analyzes their handwriting styles through a series of automated processes, regardless of what they have written. For such systems, different levels of analysis can be applied. They range from using global features extracted at the document, paragraph, and word level, such as texture, curvature, average slant, aspect ratios, entropies to much more local features extracted from an allograph, which represents the characteristic shape of a written character, or from a grapheme, which is a graphical component with no semantic meaning extracted generally with an over-segmentation method, or at even a lower microlevel. In [8], a text-independent writer approach making use of writer-dependent texture features has been presented. Specifically, Gabor filtering and grayscale co-occurrence matrices have been used for texture analysis. A 96% identification accuracy has been obtained on 1,000 testing documents from 40 subjects. In [9], morphological processing has been proposed to extract information about the geometrical and structural properties of words. Classification has been performed by using both a multilayer perceptron and a Bayesian classifier. An accuracy of 95% over a database containing only one word written by 50 writers 45 times, both in English and Greek, has been obtained. An approach for text-independent writer identification based on the use of a codebook of connected component contours has been proposed in [10] with application to uppercase Western letters. The probability density function (PDF) of the connected component contours has been used as representative feature of the writers. The performances of the method have been improved by considering, as additional feature, the distribution of local angles along the edges. Text-independent writer recognition systems can also make use of stochastic approaches like Hidden Markov Model (HMM)-based techniques, which, being robust to noise and able to cope with shape variation, represent one of the most popular strategies used to match the extracted features. In [11], HMM modeling has been performed both for writer identification and verification using static information. An identification rate of 96% has been attained when considering 8,600 text lines from the 100 writers and an error rate of 2.5% has been reported for writer verification.

In the recent past, there has been an increasing trend in proposing approaches that utilize allograph prototypes, which are different graphical representations of the characters composing an alphabet, for writer identification. Such approaches have been gaining popularity due to their simplicity and promising identification rates. The advantage of working at the character level is that a set of consistent templates can be produced to model the handwriting styles of writers. The allograph-based approaches share the following common steps. First, a set of representative templates of possible allographs of a letter has to be defined either manually or automatically. The so obtained set is used as a base where the different handwriting patterns will be projected. It results in a vector description corresponding to a distribution over the available allographs. The writer retrieval consists in matching two distributions, one coming from the reference dataset obtained during the enrollment process, the second one obtained from the questioned document. In [12], the problem of both writer identification and verification has been tackled by performing the analysis both at the texture level and at the allograph level. At the texture level, the orientation and curvature information have been encoded into a contour-based joint directional PDF. At the allograph level, portions of the character shapes, namely, graphemes, have been extracted and clustered to obtain a grapheme codebook. The PDFs of these common shapes have been then estimated and used to discriminate the writers. Furthermore, working at the character level allows visualization by forensic experts. This helps the forensic analysis of the handwriting of individuals as in [13] where dynamic time warping has been used to cluster the allographs and then build the representative vector of each writer by using the frequency of occurrence of each allograph for each character. In [14], a text-independent writer identification allograph-based approach has been presented. Specifically, 26 lowercase letters, 10 prototypes per letter, a vector description based on the frequency of both occurrency and nonoccurency of a feature in the document, and a chi-square metric have been used. An identification rate of 99.2% over a reference database generated by 120 writers has been obtained. The performance of the proposed method has been also discussed versus the choice of the letters to be used, the effect of the length of the text in the test document, the use of different matching metrics, and the method to compute the distributions. Eventually, the discriminative power of the alphabets has been analyzed.

Applications

Applications of handwriting recognition range from reading handwritten texts on postal envelopes or bank checks, to analog-to-digital conversion of textual data for automatic information indexing and retrieval.

Applications for either writer identification or writer verification can be found in several domains. The ancient manuscripts, for example, could serve to study the evolution of the style and form of writing over time, that in turn reflects the historical and cultural changes of the society. Knowledge about individual letters, ligatures, punctuation and abbreviations, and the way they have evolved enables the paleographers and historians to identify the period when a manuscript was written. Handwriting has also an interesting relationship with neuroscience and several neurological disorders that could affect the writing motor skills. Handwriting could therefore be of diagnostic value for the neurologists, particularly if they have previous writing samples of the patient [15, 16]. Another domain of interest is graphology, where handwriting is used as an insightful means of personality profiling, highlighting the character traits, and tracking the feelings and emotions of a person. As a matter of fact, graphologists claim that handwriting could reveal more than 200 personality traits including moods, temperaments, thinking patterns, fears, work drive, maturity, social interactions, and so forth. Handwriting recognition has also interesting applications in the field of forensic science where examiners have to identify the authorship of a questioned document (e.g. a will, a ransom note, or a threatening letter), verify signatures, identify forgeries, detect alterations, or analyze indented writings. At last, another application of writer identification is the identification of the writer in order to perform writer adaptation to be used in the framework of handwriting recognition so that the system is tuned to a specific writing style and, hence, can outperform a generic omni-writer system.

Recommended Reading

1. Plamondon R, Feng R, Woch A (2003) A kinematic theory of rapid human movements. Biol Cybern 89(2):126–138
2. Plamondon R, Srihari SN (2000) On-line and off-line handwriting recognition: a comprehensive survey. IEEE Trans Pattern Anal Mach Intell 22(1):63–84
3. Hochberg J, Kelly P, Thomas T (1997) Lila Kerns, Automatic script identification from document images using cluster-based templates. IEEE Trans Pattern Anal Mach Intell 19(2): 176–181
4. Busch A, Boles WW, Sridharan S (2005) Texture for script identification. IEEE Trans Pattern Anal Mach Intell 27(11):1720–1732
5. Fujisawa H (2008) Forty years of research in character and document recognition an industrial perspective. Pattern Recognit 41:2435–2446
6. Srihari SN, Cha SH, Arora H, Lee S (2002) Individuality of handwriting. J Forensic Sci 47(4):856–872
7. Tomai CI, Zhang B, Srihari SN (2004) Discriminatory power of handwritten words for writer recognition. In: 17th international conference on pattern recognition, vol 2, Cambridge, UK, pp 638–641
8. Said HES, Tan TN, Baker KD (2000) Personal identification based on handwriting. Pattern Recognit 33(1):149–160
9. Zois EN, Anastassopoulos V (2000) Morphological waveform coding for writer identification. Pattern Recognit 33(3):385–398
10. Schomaker L, Bulacu M (2004) Automatic writer identification using connected-component contours and edge-based features of uppercase western script. IEEE Trans Pattern Anal Mach Intell 26(6):787–798
11. Schlapbach A, Bunke H (2007) A writer identification and verification system using HMM based recognizers. Pattern Anal Appl 10(1):33–43
12. Bulacu M, Schomaker L (2007) Text-independent writer identification and verification using textural and allographic features. IEEE Trans Pattern Anal Mach Intell 29(4):701–717
13. Niels R, Vuurpijl L, Schomaker L (2007) Automatic allograph matching in forensic writer identification. Int J Pattern Recognit Artif Intell 21(1):61–81
14. Tan GX, Viard-Gaudin C, Kot A (2009) Automatic writer identification framework for online handwritten documents using character prototypes. Pattern Recognit 42:3313–3323
15. Eaton HD (1938) Handwriting a neurological study. Calif West Med 48(6):430–435
16. Mergl R, Juckel G, Rihl J, Henkel V, Karner M, Tigges P, Schröter A, Hegerl U (2004) Kinematical analysis of handwriting movements in depressed patients. Acta Psychiatr Scand 109(5): 383–391

Hard-Core Bit

Burt Kaliski
Office of the CTO, EMC Corporation, Hopkinton, MA, USA

Related Concepts

►One-Way Function; ►Pseudorandom Number Generator

Definition

A *hard-core bit* of a ►one-way function $y = f(x)$ is any bit or other binary function of the input x that is hard to compute significantly better than guessing given the output y alone.

Theory

Let f be a ►one-way function. According to the definition of such a function, it is difficult, given $y = f(x)$, where x is random, to recover x. However, it may be easy to determine certain information about x. For instance, the RSA function $f(x) = x^e \bmod n$ (►RSA public-key encryption) is believed to be one-way, yet it is easy to compute the ►Jacobi symbol of x, given $f(x)$:

$$\left(\frac{x^e \bmod n}{n}\right) = \left(\frac{x}{n}\right)^e = \left(\frac{x}{n}\right).$$

Another example is found in the discrete exponentiation function $f(x) = g^x \bmod p$ (discrete logarithm problem), where the least-significant bit of x is revealed from the ►Legendre symbol of $f(x)$, i.e., $f(x)^{(p-1)/2}$, which indicates whether $f(x)$ is a square and hence whether x is even.

It has therefore been of considerable interest in cryptography to understand which parts of the inverse of certain one-way functions are hardest to compute. This has led to the notion of a *hard-core bit*. Informally, a function B from inputs to {0, 1} is *hard core* with respect to f if it is infeasible to approximate $B(x)$, given $f(x)$. Here, "approximating" means predicting with probability significantly better than 1/2.

Hard-core bits have been identified for the main hard problems in public-key cryptography, including the ►discrete logarithm problem and the ►RSA problem. Blum and Micali [2] gave the first unapproximability results for the former; Alexi et al. [1], the first complete results for the latter. (See also [4] for further enhancements.)

Goldreich and Levin [3] have given a very elegant method for constructing a hard-core bit from *any* one-way function:

1. Define $g(x, r) = (f(x), r)$, where the length of r is the same as the length of x.
2. Define $B(x, r) = x_1 r_1 \oplus \cdots \oplus x_k r_k$, where x_i, r_i are the bits of x and r, and $\oplus$ is the exclusive-or operation.

If f is one-way, then g is clearly one-way. Goldreich and Levin's key result is a proof that the predicate B so constructed is hard core with respect to g. Intuitively, this can be viewed as saying that the XOR of a random subset of bits of the inverse is hard to predict.

Researchers have also studied the related problem of *simultaneous security* of multiple bits, that is, whether an individual hard-core bit remains difficult to approximate even if the values of other hard-core bits are known. Typically, some constant times log k bits can be shown to be simultaneously secure for the functions mentioned above, where k is the size of the input to the function; up to $k/2$ have been proven simultaneously secure for related functions [5]. It has been conjectured that half of the bits of the RSA/Rabin functions are simultaneously secure. See also [4] for some related results.

Applications

The primary application of hard-core bits is in constructing a ►pseudo-random number generator. As shown by Yao [6] and Blum and Micali [2], if f is a one-way permutation and B is a hard-core bit of f, then the sequence

$$B(x_0), B(x_1), B(x_2), \ldots,$$

where $x_i = f(x_{i-1})$ and x_0 is a random seed, is indistinguishable from a truly random sequence of the same length. The proof proceeds by showing that any efficient algorithm to distinguish the sequence from a truly random sequence can also be used to distinguish the supposed hard-core bit from a random value, and hence to approximate the bit.

Recommended Reading

1. Alexi WB, Chor B, Goldreich O, Schnorr C-P (1988) RSA and Rabin functions: certain parts are as hard as the whole. SIAM J Comput 17(2):194–209
2. Blum M, Micali S (1984) How to generate cryptographically strong sequences of pseudo-random bits. SIAM J Comput 13(4):850–863
3. Goldreich O, Levin L (1989) A hard-core predicate for all one-way functions. In: Proceedings of the 21st annual ACM symposium on theory of computing. ACM, New York, pp 25–32
4. Håstad J, Näslund M (2004) The security of all RSA and discrete log bits. J ACM 51(2):187–230
5. Håstad J, Schrift AW, Shamir A (1993) The discrete logarithm modulo a composite hides $O(n)$ bits. J Comput Syst Sci 47(3):376–404
6. Yao A (1982) Theory and applications of trapdoor functions. In: Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS). IEEE Computer Society Press, Los Alamitos, pp 80–91

Hardware Security Module

LAURENT SUSTEK
Inside Secure France, Atmel, France

Related Concepts

►Secure Coprocessor

Background

Security devices play an essential role in our everyday life, ensuring security for the financial transactions can be made directly or indirectly. These devices provide the ability to make transactions within a distributed and virtual environment, satisfying the request of Trust and Privacy.

One of the commonly used trusted token is the Smart Card.

When you want to pay for some gift, the vendor wants to be assured that you are the real owner – not a hacker collecting credit card numbers – of the banking account. On the other side, you need to know that the vendor is really who he claims to be before paying. Both parties in this transaction need to authenticate each other's identity. In addition, depending of the transaction's value, both parties may want the exchanges protected against hacker spying, disclosing and intercepting the exchanges of the transaction. Both parties need confidentiality and integrity.

Security technology can be divided into two types: software defined and hardware achieved.

Security Processors are computational devices that are used to execute security functions in a short execution time. In the context of Information Technology, these Security Processors have to execute trustfully these operations whether the environment is hostile or not, so they are usually integrated into a tamper-resistant device or package. These devices are known by a variety of names, including:

- Tamper-Resistant Security Module (TRSM)
- Network Security Processor (NSP)
- Host/Hardware Security Module (HSM)

An HSM is a physically secure, ►tamper-resistant security server that provides cryptographic functions to secure transactions in retail and financial applications. This includes:

- PIN (►Personal Identification Number) encryption and verification
- Debit card validation
- Stored value card issuing and processing
- Chip card issuing and processing
- Message ►authentication (►MAC Algorithms)
- Symmetric key management

With a DSP-RSA Module, the HSM can also support public key cryptographic operations including digital signatures, ►certificates, and asymmetric key management.

Acting as a peripheral to a host computer, the HSM provides the cryptographic facilities needed to implement a wide range of data security tasks.

Banks, corporations, and probably some branches of the military are using HSMs as part of their security chain.

Theory

Hardware Security Modules offer a higher level of security than software. They are normally evaluated by third parties, such as the USA's "National Institute of Standards and Technology" (NIST), through the Federal Information Processing Standards Publication (FIPS PUB 140) or French "Direction Centrale de la Sécurité des Systmés d'Information" (DCSSI). This level of security is required by some highly secured web applications, ►Public Key Infrastructures, and ►Certification Authorities.

Hardware Security Modules perform cryptographic operations, protected by hardware. These operations may include:

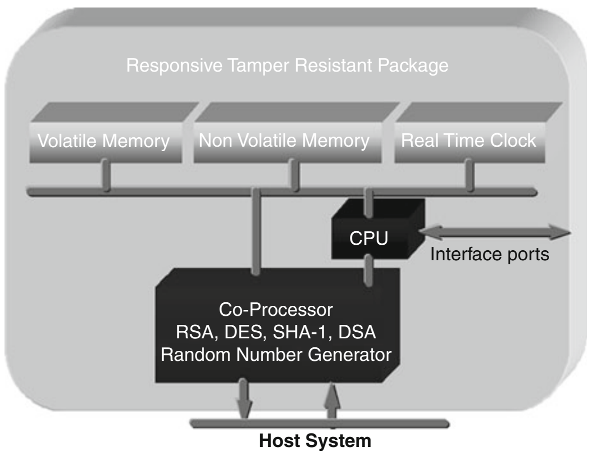
- Random number generation
- Key generation (asymmetric and symmetric) (►Asymmetric Cryptosystem and ►Symmetric Crypto system)
- Asymmetric private key storage while providing protection (security) from attack (i.e., no unencrypted private keys in software or memory):
  - Private keys used for signing and decryption
  - Private keys used in PKI for storing Root Keys

The diagram below describes the internal elements of an HSM (Fig. 1).

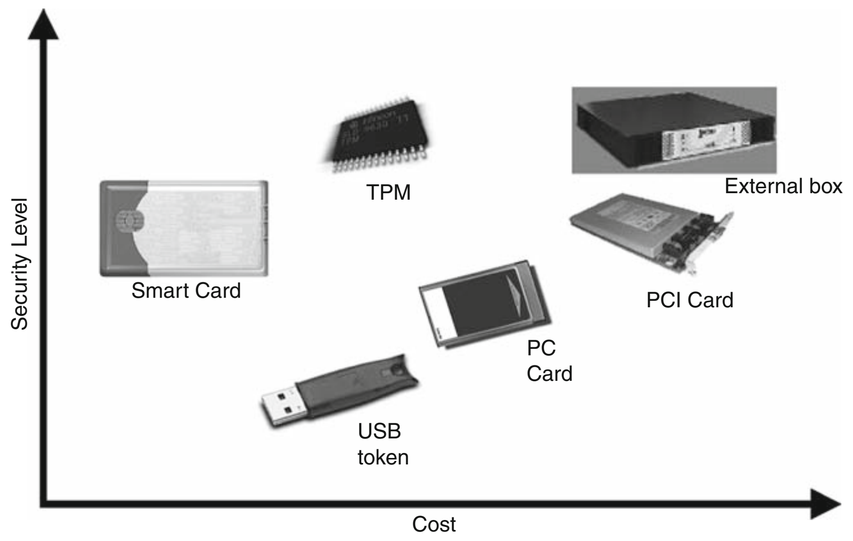
The diagram below illustrates various hardware encryption technologies and their respective positions of cost relative to security (Fig. 2). An HSM is a key element in the security chain of a system. It provides all necessary cryptographic functions in association to a secured key management for generation, storage, and handling. Its cost may mean that only companies can afford to use such devices. Overall benefit is the third-party evaluation of this piece of hardware and software, through governmental certification schemes.

Useful Links

- More on HSM: http://www.cren.net/crenca/onepagers/additionalhsm.html
- "Building a High-Performance, Programmable Secure CoProcessor" by Sean W. Smith and Steve Weingart



Hardware Security Module. Fig. 1



Hardware Security Module. Fig. 2

Hardware Security Module. Table 1

FIPS140-2	
Level	**Description**
1	Lowest security level. Basic hardware or software based cryptographic functions implementation
2	Both software and hardware implementations can meet this level. Hardware must incorporate a limited degree of tamper-evident design or employ locks to secure sensitive information. Role-based authentication is required to authorize a defined set of services. Role-based authentication required. Meets ISO15408 Common Criteria (CC) Evaluation Assurance Level EAL2 or higher
3	Physical tamper resistance is required. Indentity-based authentication (indentity check+acces autorization). Physical or logical communication interfaces separation, introducing notion of trusted path. Meets CC EAL3 level or higher
4	Hardware certified at this level must resist the most sophisticated attacks. Cryptographic functions are performed within an "envelope" protected by the security parameters. The intent of Level 4 protection is to detect a penetration of the device from any direction. Such devices are suitable for operation in physically unprotected environments. Meets CC EAL4 level or higher

- Cryptographic equipments list: http://www.ssi.gouv.fr/fr/reglementation/liste_entr/index.html
- FIPS PUB 140-2 Security Requirements for Cryptographic Modules:
  - http://csrc.nist.gov/cryptval/
  - http://csrc.nist.gov/cryptval/140-2.htm
  - PKCS #11 – Cryptographic Token Interface Standard: http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html

Definitions (Extracted from ISO 15408)

Tamper Evidence Requirement

A device that claims Tamper Evidence characteristics *shall* be designed and constructed as follows:

- *Substitution*: To protect against substitution with a forged or compromised device, a device is designed so that it is not practical for an attacker to construct a duplicate from commercially available components that can reasonably be mistaken for a genuine device.
- *Penetration*: To ensure that penetration of an SCD is detected, the device *shall* be so designed and constructed that any successful penetration *shall* require that the device be subject to physical damage or prolonged absence from its authorized location such that the device cannot be placed back into service without a high probability of detection by a knowledgeable observer.

Tamper Resistance Requirements

- *Penetration*: An SCD *shall* be protected against penetration by being Tamper Resistant to such a degree that its passive resistance is sufficient to make penetration infeasible both in its intended environment and when taken to a specialized facility where it would be subjected to penetration attempts by specialized equipment.

- *Modification*: The unauthorized modification of any key or other sensitive data stored within an SCD, or the placing within the device of a tap, e.g., active, passive, radio, etc., to record such sensitive data, *shall* not be possible unless the device be taken to a specialized facility and this facility be subjected to damage such that the device is rendered inoperable.
- *Monitoring*: Monitoring *shall* be countered by using tamper-resistant device characteristics. The passive physical barriers *shall* include the following:
  - Sheilding against electromagnetic emissions in all frequencies in which sensitive information could be feasibly disclosed by monitoring the device.
  - Privacy shelding such that during normal operation, keys pressed will not be easily observable to other persons. (For example, the device could be designed and installed so that the device can be picked up and shielded from monitoring by the user's own body.)

Where parts of the device cannot be appropriately protected from monitoring; these parts of the device *shall* not store, transmit, or process sensitive data. The device *shall* be designed and constructed in such a way that any unauthorized additions to the device, intended to monitor it for sensitive data, *shall* have a high probability of being detected before such monitoring can occur.

- Removal: If protection against removal is required, the device *shall* be secured in such a manner that it is not economically feasible to remove the device from its intended place of operation.

Hash Agility

David Challener
Applied Physics Laboratory, Johns Hopkins University, Laurel, MD, USA

Related Concepts

►MD4; ►MD5; ►RIPEMD-160; ►SHA-1; ►SHA-2; ►TPM

Definition

In the broadest sense, Hash Agility refers to the ability of a program, algorithm, protocol, or device to use any of a group of cryptographic hash algorithms in performing its task. In a client server protocol, hash agility also requires that two (or more) entities be able to negotiate a hash algorithm or algorithms that satisfies both the security requirements of all entities and also the abilities of all entities [1].

Background

Many of the popular Hash algorithms are susceptible to cryptanalytic attacks, especially MD 4 and 5 and SHA-1 (to a lesser degree) [7, 8]. As a result, programs, protocols, and devices have been changed in order to use different (and hopefully more secure) hash algorithms. NIST is currently sponsoring a contest similar to the contest that resulted in AES to create the next secure hash algorithm. It is likely that when this new hash algorithm is selected that it will be adopted quickly. The contest is not due to be finished until 2012, so specifications and designs produced today are being created to be adaptable, so they can take advantage of the new hash algorithm without major changes.

Hash agility provides for backwards compatibility with programs that have been upgraded to the new hash algorithms in use, but often it provides an excuse for programs to not upgrade.

Theory and Applications

Protocols and Applications

- Transport Layer Security (TLS) (can use MD5 or SHA-1)
- Internet Protocol Security (IPsec) (can use MD5, SHA-1, SHA-256, SHA-384, and SHA-512)
- Secure Sockets Layer (SSL) (can use MD5, SHA-1, SHA-224, SHA-256, SHA-384, and SHA512)
- Pretty Good Privacy (PGP) (can use MD5, RIPEMD-160, SHA-1, SHA-256, SHA-512)
- GPG (can use MD5, SHA-1, SHA-256, SHA-512)
- GnuPG (can use MD5, RIPEMD-160, SHA-1, SHA-256, SHA-512)
- Secure Shell (SSH) (can use MD5 or SHA-1, SHA-256, SHA-384, and SHA-512)
- Secure / Multipurpose Internet Mail Extensions (S/MIME) (can use MD5, SHA-1, SHA-224, SHA-256, SHA-384, and SHA512))

Devices

Hardware devices are typically designed to support only a single hash algorithm. This is because silicon is expensive, and most software that talks to the hardware device is able to support multiple hash algorithms. The TPM 1.1b and 1.2 only support SHA-1. The Common Access Card used by the US Military only supports SHA-1. The new PIV-2 cards which replace the CAC cards will provide support for SHA-1/224 and 256. Syprus makes smartcards and USB sticks that can handle SHA-1/224/256/384/512.

TPMs and Hash Agility

The TPM Specifications 1.1b and 1.2 [5] were written with SHA-1 ingrained throughout the structures and functions. When attacks on SHA-1 surfaced which could lead to collisions (the ability to find two inputs with the same output), there was immediate concern. Although there was no known way to exploit a collision attack against the TPM specification, it is a known fact in cryptography that attacks only get worse.

Two implicit problems in the TPM were the SHA-1 dependent key structures and the SHA-1 extended Platform Control Registers (PCRs). The TPM uses 2048 bit RSA keys to store other 2048 bit RSA keys (using SHA-1 hashes for HMAC keys and integrity settings associated with the stored RSA key), and the PCRs are used to store measurements by extending them with SHA-1 into memory locations of 20 bytes. A less obvious problem occurs when one wants to develop a chain of trust for TPM operations.

Key Storage

The keys in the TPM specification are stored encrypted with 2048 bit RSA keys, known as Storage Keys. Normally a 2048 bit key does not have enough space to encrypt another 2048 bit key in a single encryption, but the specification gets around this by only storing one of the two primes in the encrypted blob, and attaching the public key (unencrypted) which includes the product of the two primes. Once the first prime is decrypted inside the TPM, a simple division is used from the public key to obtain the other prime. Because of structure tagging, this leaves 1008 bits or 124 bytes in which to store other information. However there is a lot of other information to store, including: Use authorization (20 bytes, corresponding to one SHA-1 hash), Migration authorization (another 20 bytes corresponding to a SHA-1 hash), Public Data Digest (another 20 bytes from a SHA-1 hash), Payload (1 byte), Random Number for OAEP (another 20 bytes). Other various things take up most of the remaining 40 bytes, but even without them, increasing the hash from 20 bytes to 32 byte (moving from SHA-1 to SHA-256) would exceed the available space.

It is clear that when the new TPM specification appears, key structures will have to change in order to accommodate larger hashes [6]. The specification is expected to be hash agile, that is, the specification will be written in such a way that it will be possible to be compliant to the specification while using any hash algorithm. Acceptable hash algorithms will be chosen by users of the specification. However such changes will likely be done in a way that allows maintaining of backwards compatibility [5].

In order to allow for flexible algorithm use, it is likely this will require an additional, two step, asymmetric/symmetric means of encrypting keys stored outside the TPM. This technique uses the asymmetric encryption to encrypt a symmetric key which in turn is used to encrypt the next key. Since symmetric encryption is not limited in space, any size hash will be usable.

Platform Control Registers

PCRs work by a process called extension. If data is extended into a PCR, the data in the PCR is replaced with Hash (old data/new data), where / means concatenation. This is a one way function, assuming the hash has not been broken. The 1.1b and 1.2 specification reserve 20 byte registers for use as PCRs (16 of them in 1.1b and 24 in 1.2). In order to be hash agile, this will have to change, although it is not clear what technique will be used.

Obvious problems that need to be solved include the size of registers set aside for the hash algorithm, whether multiple types of hashes can be used simultaneously in different registers, and how data extended into PCRs representative of a boot sequence may be simultaneously represented via different hash algorithms.

Certificate Chains

In the current TPM design, the chain of trust starts with a certificate for the TPM and for the platform on which the platform is attached. These two certificates are used to create a certificate for an Attestation Identity Key (AIK) which in turn is used to provide a certificate for other non-migratable keys on the platform. The AIK can also be used to attest to values of Platform Control Registers. In the 1.1 and 1.2 design of the TPM, the certificates are all implicitly created using SHA-1. In the future, when multiple hashes can be used, it becomes important that the hash algorithm used in the certificates be explicit, and also not be spoofable by using a weak hash algorithm to attest to the hash algorithm being used.

Recommended Reading

1. Kaliski Jr BS (2002) On hash function firewalls in signature schemes. In: Preneel B (ed) Topics in Cryptology, LCNS, vol 2271, pp 1–16
2. NIST (2008) Cryptographic hash project, December 2008. At http://csrc.nist.gov/groups/ST/hash/index.html
3. NIST (2008) Fips 180-3. October 2008. At http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf
4. Schneier B (1996) Applied cryptography: protocols, algorithms, and source code in C, 2nd edn.Wiley, New York
5. Trusted Computing Group (2006) Tpm main specification

6. Trusted Computing Group (2009) Features under consideration for tpm next, 2009. At http://www.trustedcomputinggroup.org/files/resource_files/0CD79678-1D09-3519-ADDAFD2ED5450D0A/Features%20Under%20Consideration%20for%20TPM%20next%20(FINAL).pdf
7. Yin YL, Wang X, Yu H (2005) Efficient collision search attacks on sha-0. In: Advances in Cryptology–Crypto 2005, LCNS vol 3621. Springer, Heidelberg
8. Yin YL, Wang X, Yu H (2005) Finding collisions in the full sha-1. In: Advances in Cryptology–Crypto 2005. Springer, Heidelberg

Hash-Based Signatures

Tanja Lange
Coding Theory and Cryptology, Eindhoven Institute for the Protection of Systems and Information, Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven, The Netherlands

Synonyms

Lamport one-time signatures; Merkle-hash-trees signatures; Winternitz one-time signatures

Related Concepts

►Hash Functions; ►Post-quantum Cryptography; ►Signature

Definition

Hash-based signature schemes are public key signatures that are based on the one-wayness of cryptographic ►hash functions.

Theory

The first hash-based signature scheme is Lamport's one-time signature scheme [6] (see also [3], p. 650). To sign messages of length k the system is set up as follows:

Let H be a one-way function $H : \{0,1\}^k \to \{0,1\}^k$. The signer chooses $2k$ random strings $x_1[0], x_1[1], \ldots, x_k[0], x_k[1]$ and stores them as his secret key X. Then he computes $y_i[b] = H(x_i[b])$ for $b \in \{0,1\}$ and $1 \le i \le k$. The public key is the vector $Y = (y_1[0], y_1[1], \ldots, y_k[0], y_k[1])$ of $2k$ strings of length k each.

The signature $S(X, m)$ of a message m is $x_1[m_1], \ldots, x_k[m_k]$, where $m = (m_1, \ldots, m_k)$. The verifier checks whether $H(x_i[m_i]) = y_i[m_i]$ for $1 \le i \le k$.

The scheme is called *one-time signature* because a public key can be used only for one signature: if a second message is signed that differs from the first in at least two hash bits then an attacker can generate further valid signatures. Assume that the first message satisfies $m_1 = (0, 1, 1, \ldots)$ and the second message satisfies $m_2 = (1, 1, 0, \ldots)$. Then $x_1[0], x_1[1], x_2[1], x_3[0], x_3[1], \ldots$ are revealed, allowing anybody to sign also messages $(1, 1, 1, \ldots)$ or $(0, 1, 0, \ldots)$, where the values in the dots need to match one of the previously signed messages.

To extend this one-time signature scheme to longer messages let G be a cryptographic hash function, $G: \{0,1\}^* \to \{0,1\}^k$. It is possible, and commonly done, to choose H as G restricted to input of length k. Setting up the public key works as stated above.

The signature $S(X, r, m)$ of a message m is $r, x_1[g_1], \ldots, x_k[g_k]$, where $G(r, m) = (g_1, \ldots, g_k)$ and r is a random string. The verifier computes $G(r, m)$ and verifies whether $H(x_i[g_i]) = y_i[g_i]$ for $1 <= i <= k$.

Each public key consists of $2k^2$ bits, and the secret keys have the same size. It is possible to save on storage at the signer's side by generating the $x_i[b]$ as outputs of a ►pseudorandom number generator so that only the seed needs to be stored but this does not change the size of the public key. The main drawback is that each extra signature needs new public values and that these values must be generated and made public long before the signature is generated.

In 1979, Merkle extended the scheme to handle multiple signatures [7]. Let $H : \{0,1\}^* \to \{0,1\}^k$ be a cryptographic hash function. The number of signatures needs to be fixed beforehand, for example, user Alice prepares the system so that she can sign up to eight messages.

For this she computes eight Lamport one-time public keys $Y_1, Y_2, \ldots, Y_8$ with corresponding private keys $X_1, X_2, \ldots, X_8$, i.e., $X_i = (x_{i,1}[0], x_{i,1}[1], \ldots, x_{i,k}[0], x_{i,k}[1])$ $Y_i = (y_{i,1}[0], y_{i,1}[1], \ldots, y_{i,k}[0], y_{i,k}[1])$ and $y_{i,j}[b] = H(x_{i,j}[b])$. Each of the X_i and the Y_i has $2k^2$ bits. The public keys are then arranged in a binary tree as follows: the eight public keys are the leaves of the tree. Values of neighboring nodes are hashed together to form the next layer of the tree, e.g., Y_1 and Y_2 form $Y_9 = H(Y_1, Y_2) = H((y_{1,1}[0], y_{1,1}[1], \ldots, y_{1,k}[0], y_{1,k}[1]), (y_{2,1}[0], y_{2,1}[1], \ldots, y_{2,k}[0], y_{2,k}[1]))$. The process continues to the root as shown in the first part of Fig. 1.

The arrows indicate that computing values closer to the root requires knowledge of the values on the layer above. The public key in Merkle's signature scheme is Y_{15}.

To sign the first message m_1, Alice generates some randomness r_1, computes $S(X_1, r_1, m_1)$ as in Lamport's scheme, and also includes Y_1, Y_2, Y_{10}, and Y_{14} as the authentication path to Y_{15}.

The receiver hashes the entries of X_1 given by $S(X_1, r_1, m_1)$ and verifies that they match the corresponding entries of Y_1. If so the receiver computes $Y_9 = H(Y_1, Y_2)$, then computes $Y_{13} = H(Y_9, Y_{10})$ by hashing the computed Y_9 together with Y_{10} (given as part of the signature), and

$$\begin{tikzcd}[column sep=tiny]
X_1 \arrow[d] & & X_2 \arrow[d] & & X_3 \arrow[d] & & X_4 \arrow[d] & & X_5 \arrow[d] & & X_6 \arrow[d] & & X_7 \arrow[d] & & X_8 \arrow[d] \\
Y_1 \arrow[dr] & & Y_2 \arrow[dl] & & Y_3 \arrow[dr] & & Y_4 \arrow[dl] & & Y_5 \arrow[dr] & & Y_6 \arrow[dl] & & Y_7 \arrow[dr] & & Y_8 \arrow[dl] \\
& Y_9 = H(Y_1, Y_2) \arrow[drr] & & & & Y_{10} = H(Y_3, Y_4) \arrow[dll] & & & & Y_{11} = H(Y_5, Y_6) \arrow[drr] & & & & Y_{12} = H(Y_7, Y_8) \arrow[dll] & \\
& & & Y_{13} = H(Y_9, Y_{10}) \arrow[drrrr] & & & & & & & & Y_{14} = H(Y_{11}, Y_{12}) \arrow[dllll] & & & \\
& & & & & & & Y_{15} = H(Y_{13}, Y_{14}) & & & & & & &
\end{tikzcd}$$

$$\begin{tikzcd}[column sep=tiny]
\boxed{X_1} \arrow[d] & & X_2 \arrow[d] & & X_3 \arrow[d] & & X_4 \arrow[d] & & X_5 \arrow[d] & & X_6 \arrow[d] & & X_7 \arrow[d] & & X_8 \arrow[d] \\
\boxed{Y_1} \arrow[dr] & & \boxed{Y_2} \arrow[dl] & & Y_3 \arrow[dr] & & Y_4 \arrow[dl] & & Y_5 \arrow[dr] & & Y_6 \arrow[dl] & & Y_7 \arrow[dr] & & Y_8 \arrow[dl] \\
& Y_9 = H(Y_1, Y_2) \arrow[drr] & & & & \boxed{Y_{10} = H(Y_3, Y_4)} \arrow[dll] & & & & Y_{11} = H(Y_5, Y_6) \arrow[drr] & & & & Y_{12} = H(Y_7, Y_8) \arrow[dll] & \\
& & & Y_{13} = H(Y_9, Y_{10}) \arrow[drrrr] & & & & & & & & \boxed{Y_{14} = H(Y_{11}, Y_{12})} \arrow[dllll] & & & \\
& & & & & & & Y_{15} = H(Y_{13}, Y_{14}) & & & & & & &
\end{tikzcd}$$

Hash-Based Signatures. Fig. 1 Merkle tree for 8 signatures and authentication path for first signature

then finally hashes Y_{13} and Y_{14} together and accepts the signature if the computed value for $H(Y_{13}, Y_{14})$ matches the public key Y_{15}.

Similarly, the second message is signed by sending $S(X_2, r_2, m_2), Y_1, Y_2, Y_{10}, Y_{14}$, the third one by $S(X_3, r_3, m_3)$, Y_3, Y_4, Y_9, Y_{14}, etc.

This method reduces the size of the public key to k bits, independent of how many messages the tree covers, but the secret key has size $2k^2 2^s$ if the tree is set up to sign 2^s messages, and a signature has size $5k^2 + (s-1)k$ if the tree is set up to sign 2^s messages. The latter can be reduced to $3k^2 + sk$ by introducing another layer of hashing at the leaf level: let $Z_i = H(Y_i)$ and replace all Y_i in the tree by Z_i; the first signature in the example is then given by $S(X_1, r_1, m_1), Y_1, Z_2, Z_{10}, Z_{14}$ and needs $3k^2 + 3k$.

Storage for the secret key increases if the Y_i are not recomputed as part of signature generation. Several papers study the costs of tree traversal to minimize the time needed in signature generation. A comprehensive summary is given in [2].

The Lamport one-time signature can be replaced by a Winternitz one-time signature, which is more compact – signing w bits with a single preimage $x \in \{0,1\}^k$ for some choice of $w < k$ – but needs up to 2^{w-1} executions of H per w bits. For details see [4].

If the tree grows so that many signatures can be generated for one public key the time to compute the authentication path grows and makes the scheme prohibitively slow. Tree chaining solves this issue by using Merkle hash trees on different levels. The tree on the lowest level has as its root the public key, its leaves are used to sign the roots of trees in higher levels, etc. Only the trees on the top level are used to sign messages. This has the advantage that only the smaller trees along the authentication path to the public key need to be traversed and not the entire union of all trees. This approach can be combined with using pseudorandom number generators to generate the secret keys for all subtrees. For details see [2] and the references therein.

Applications

The security of most hash functions is not affected by Shor's algorithm [8]; thus when large quantum computers become reality, the security of systems based on hash functions is only affected by Grover's algorithm [5], which in essence means that preimages of a hash function with output size k bits can be found in $2^{k/2}$ instead of in 2^k

operations. The impact of Grover's algorithm on collision security depends on the exact model of computation and is not as large; see [1] and the references therein.

This means that doubling the size of the hash function is sufficient to protect hash-based signature schemes against attacks using quantum computers. This is in sharp contrast to the ►RSA digital signature scheme and ►elliptic curve signature schemes, which are broken by quantum computers. Hash-based signatures are studied as part of ►post-quantum cryptography.

Recommended Reading

1. Bernstein DJ (2009) Cost analysis of hash collisions: will quantum computers make SHARCS obsolete? In: Workshop record of SHARCS 2009: special-purpose hardware for attacking cryptographic systems, Lausanne, 9–10 Sep 2009
2. Buchmann J, Dahmen E, Szydlo M (2009) Hash-based digital signature schemes. In: Bernstein DJ, Buchmann J, Dahmen E (eds) Postquantum cryptography. Springer, Berlin, pp 35–93. http://www.cdc.informatik.tu-darmstadt.de/~dahmen/papers/hashbasedcrypto.pdf
3. Diffie W, Hellman M (1976) New directions in cryptography. IEEE Trans Inf Theory 22:644–654. http://www-ee.stanford.edu/~hellman/publications/24.pdf
4. Dods C, Smart NP, Stam M (2005) Hash based digital signature schemes. In: Smart NP (ed) Cryptography and coding, 10th IMA international conference, proceedings Cirencester, 19–21 Dec 2005. Lecture notes in computer science, vol 3796. Springer, Heidelberg, pp 96–115
5. Grover LK (1996) A fast quantum mechanical algorithm for database search. In: STOC, Philadelphia, pp 212–219
6. Lamport L (1979) Constructing digital signatures from a one way function. Technical Report SRI-CSL-98, SRI International Computer Science Laboratory
7. Merkle RC (1979) Secrecy, authentication, and public key systems. PhD thesis, Stanford University. http://www.merkle.com/papers/Thesis1979.pdf
8. Shor PW (1997) Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J Comput 26(5):1484–1509

Hash Chain

Dwight Horne
Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX, USA

Synonyms

One-Way Chain

Related Concepts

►Challenge-Response Authentication; ►Cryptographic Hash Function

Definition

A hash chain is a sequence of values derived via consecutive applications of a cryptographic hash function to an initial input. Due to the properties of the hash function, it is relatively easy to calculate successive values in the chain but given a particular value, it is infeasible to determine the previous value.

Background

Leslie Lamport originally proposed the use of hash chains to provide a secure means of authentication using ►one-time passwords when the attacker is able to ►eavesdrop on communications [1]. Haller later described a prototype authentication system for UNIX systems based on Lamport's scheme [2]. Researchers have subsequently identified a number of contexts in which hash chains are beneficial such as network routing, micropayment schemes, and efficient authentication in various circumstances.

Theory

The utility of the hash chain stems primarily from the following properties of the cryptographic hash function $H(x)$:

1) H accepts a variable size input and computes a fixed size output.
2) $H(x)$ is "easy" to compute.
3) It is computationally infeasible to invert $H(x)$ (i.e., it is a ►one-way function).
4) Given x and $H(x)$, it is computationally infeasible to find another value y such that $H(x) = H(y)$.
5) It is computationally infeasible to find any two values x and y such that $H(x) = H(y)$.

In the second property, "easy" is not often rigorously defined in the literature but it may be thought of as polynomial-time in terms of ►computational complexity. Alternatively, in a practical sense, "easy" means that it should be feasible to produce efficient implementations in both software and hardware. This property is particularly important when used in power-constrained environments such as with applications of hash chains in wireless sensor networks. The third property is often referred to as the one-way property (►One-Way Function). It is essential that given an output y, one cannot feasibly find x such that $H(x) = y$. The fourth and fifth properties are often referred to as weak collision resistance and strong collision resistance, respectively (►Collision Resistance).

A hash chain of n values is often denoted $H^n(x)$ where the i^{th} value in the chain would be computed as $x_i =$

$H(x_{i-1})$. Hence, if the initial ►password were x then

$$H^2(x) = H(x), H(H(x))$$
$$H^3(x) = H(x), H(H(x)), H(H(H(x)))$$

and so on. Note that, given a value in the chain x_i, one cannot feasibly determine the previous value x_{i-1} due to the properties of the hash function. However, if x_{i-1} were subsequently supplied then one may compute to verify that $H(x_{i-1}) = x_i$ and feel confident that the individual that supplied the value knows the hash chain and thus, the initial value or password.

Applications

Lamport (1981) first proposed hash chains for use with computer system authentication. When used in the context of authentication, the values in the hash chain are supplied in reverse order as one-time passwords. Initially, the system can either compute the entire hash chain or it can be supplied with the last value in the chain, x_n. Note that the system need only remember the last successfully used password in this case.

For instance, suppose the user supplied y for the first login. To authenticate the user, the system would simply compute $H(y)$ to see if the result is equal to x_n. If so, the user correctly supplied the previous value in the hash chain (i.e., $y = x_{n-1}$) and she has been authenticated. In this example, if an attacker were to intercept y or compromise the system to discover x_n, the attacker could not feasibly determine the next valid password (i.e., the previous value in the hash chain) due to the one-way property of the hash function. The collision resistance properties of the hash function are also essential for practical applications of hash chains. In the case of authentication, an attacker that knew the last-used password x_i, but did not know x_{i-1} or x, could supply another string z such that $H(z) = H(x_{i-1}) = x_i$ in the absence of collision resistance.

After Lamport's proposal of utilizing hash chains for computer system authentication, hash chains quickly became an important cryptographic primitive. Hash chains have subsequently been applied in a variety of contexts such as for ►secure routing protocols [3], micro-payments [4], certificate revocation in public key cryptosystems (►Public Key Infrastructure) [5], forward security (►Perfect Forward Security) and privacy protection with Radio Frequency Identification (RFID) systems (►RFID Security) [6], as well as efficient authentication with address resolution [7], system event logs [8], multicast network traffic [9], and wireless sensor networks [10].

Recommended Reading

1. Lamport L (1981) Password authentication with insecure communication. Commun ACM 24(11):770–772
2. Haller N (1994) The S/KEY one-time password system. In: Proceedings of the Symposium on Network and Distributed Systems Security. pp 151–157
3. Hauser R, Przygienda T, Tsudik G (1997) Reducing the cost of security in link-state routing. In: Proceedings of the Symposium on Network and Distributed Systems Security. pp 93–99
4. Rivest R, Shamir A (1996) PayWord and MicroMint: two simple micropayment schemes. RSA Laboratories, Redwood City, CA
5. Micali S (1996) Efficient certificate revocation. Technical Memo MIT/LCS/TM-542b. Laboratory for Computer Science, Massachusetts Institute of Technology US Patent 5,666,416,9 Sep 1997
6. Ohkubo M, Suzuki K, Kinoshita S (2003) Cryptographic approach to privacy-friendly tags. In: RFID Privacy Workshop. MIT, Massachusetts
7. Goyal V, Tripathy R (2005) An efficient solution to the ARP cache poisoning problem. In: Boyd C, Gonzaìlez Nieto JM (eds) Proceedings of the 10th Australian Conference on Information Security and Privacy, Brisbane, 4–6 July 2005. Lecture Notes in Computer Science (LNCS), vol 3574. Springer-Verlag, Berlin, pp 40–51
8. Schneier B, Kelsey J (1998) Cryptogaphic support for secure logs on untrusted machines. In: Proceedings of the 7th USENIX Security Symposium, San Antonio, 26–29 January 1998. USENIX Press, Berkeley, pp 53–62
9. Perrig A, Canetti R, Song D, Tyger JD (2000) Efficient authentication and signing of multicast streams over lossy channels. In: Proceedings of the IEEE Symposium on Security and Privacy, Berkeley, 14–17 May 2000. IEEE Computer Society, Los Alamitos
10. Perrig A, Szewczyk R, Wen V, Culler D, Tygar JD (2001) SPINS: Security protocols for sensor networks. In: Proceedings of the 7th Annual ACM International Conference on Mobile Computing and Networks, Rome, 16–21 July 2001. ACM, New York, pp 189–199

Hash Functions

Bart Preneel
Department of Electrical Engineering-ESAT/COSIC, Katholieke Universiteit Leuven and IBBT, Leuven-Heverlee, Belgium

Introduction

Cryptographic hash functions take input strings of arbitrary (or very large) length and map these to short fixed length output strings. The term *hash functions* originates from computer science, where it denotes a function that compresses a string of arbitrary length to a string of fixed length. Hash functions are used to allocate as uniformly as possible storage for the records of a file. For cryptographic applications, we distinguish between unkeyed and keyed hash functions. We consider here only cryptographic hash functions without a secret parameter or secret key; these are also known as *Manipulation Detection Codes* (or *MDCs*). An important class of keyed hash functions are ►MAC Algorithms, which are used for information authentication. They can reduce the authenticity of a large quantity of information to the secrecy and authenticity of a short secret key.

Unkeyed cryptographic hash functions or MDCs will in the remainder of this entry be called hash functions. These functions can also provide information authentication in a natural way: One replaces the protection of the authenticity of the original input string by the protection of the authenticity of the short hash result. A simple example of such a process is the communication of a large data file over an insecure channel. One can protect the authenticity of this data file by sending the hash result of the file over an authenticated channel, e.g., by reading it over the phone or by sending it by regular mail or telefax.

The most common application of hash functions is in ►digital signatures: One will apply the signing algorithm to the hash result rather than to the original message; this brings both performance and security benefits. With hash functions, one can also compare two values without revealing them or without storing the reference value in the clear. The typical examples are passwords and passphrases: The verifier will store the image of the passphrase under a hash function; on receipt of the passphrase, it will apply the hash function and check whether the result is equal to this image. Hash functions can also be used to commit to a value without revealing it. The availability of efficient hash functions has resulted in their use as ►pseudorandom functions (for key derivation), and as building blocks for ►MAC algorithms (e.g., ►HMAC), ►block ciphers (e.g., Bear and Lion [5] and Shacal [41]) and ►stream ciphers (e.g., ►SEAL). Finally, many results in cryptology rely on the ►random oracle model: Informally one assumes the existence of a random function that can be queried on arbitrary inputs to yield a random output. If one needs to instantiate a random oracle in practice, one typically uses a hash function.

Definitions: It will be assumed that the description of the hash function h is publicly known; one also requires that given the inputs, the computation of the hash result must be *efficient*.

One-Way Hash Function (OWHF)

The concept of one-way hash functions was introduced by Diffie and Hellman in [25]. The first informal definition was given by Merkle [52, 53] and Rabin [68].

Definition 1 A one-way hash function (OWHF) *is a function h satisfying the following conditions:*

1. *The argument X can be of arbitrary length and the result $h(X)$ has a fixed length of n bits.*
2. *The hash function must be* one-way *in the sense that given a Y in the image of h, it is computationally infeasible to find a message X such that $h(X) = Y$ (►preimage resistant) and given X and $h(X)$ it is computationally infeasible to find a message $X' \neq X$ such that $h(X') = h(X)$ (►second preimage resistant).*

Typical values for the length n of the hash result are 64. . .128. A function that is preimage resistant is known as a ►one-way function (but preimage resistance is typically used for hash functions). It is clear that for permutations or injective functions, second preimage resistance is not relevant. Note that some authors call second preimage resistance as *weak collision resistance*. For some applications (e.g., ►pseudorandom functions and ►MAC algorithms based on hash functions), a large part of the input of the hash function may be known, yet one requires that it is hard to recover the unknown part of the input. This property is known as *partial preimage resistance*.

Collision Resistant Hash Function (CRHF)

The importance of ►collision resistance for hash functions used in digital signature schemes was first pointed out by Yuval [83]. The first formal definition of a CRHF was given by Damgård [19, 20]. An informal definition was given by Merkle [53].

Definition 2 A collision resistant hash function (CRHF) *is a function h satisfying the following conditions:*

1. *The argument X can be of* arbitrary *length and the result $h(X)$ has a* fixed *length of n bits.*
2. *The hash function must be an OWHF,* i.e., *preimage resistance and second preimage resistant.*
3. *The hash function must be* collision resistant*: it is computationally infeasible to find two distinct messages that hash to the same result.*

Note that one finds in the literature also the terms *collision freeness* and *collision intractible*.

Relation Between Definitions

It is clear that finding a second preimage cannot be easier than finding a collision: Therefore, the second preimage condition in the definition of a CRHF seems redundant. However, establishing the exact relation between these conditions requires formal definitions. Under certain conditions, detailed in [74], collision resistance implies both second preimage resistance and preimage resistance. A formalization of collision resistance requires a public parameter (also called a ▶key), even if in practice one uses a fixed function. If such a parameter is also introduced for preimage and second preimage resistance, one has then the choice between randomizing the challenge (for the second preimage or preimage attack), the key, or both. The relationship between these definitions is studied by Rogaway and Shrimpton [74]. Earlier work on this topic can be found in [21, 37, 62, 78, 85].

A ▶universal one-way hash function (UOWHF) was defined by Naor and Yung [59]. It is a class of functions indexed by a public parameter (called a key), for which finding a second preimage is hard; the function instance (or parameter) is chosen after the challenge input, which means that finding collisions for a particular instance does not help an attacker. It corresponds to one of the cases considered by Rogaway and Shrimpton.

Simon [77] provides a motivation to treat collision resistant hash functions as independent cryptographic primitives. He shows that no provable construction of a CRHF can exist based on a black-box one-way permutation, i.e., a one-way permutation treated as an oracle.

One may need other properties of hash functions: e.g., when hash functions are combined with a multiplicative ▶digital signature scheme (e.g., the plain ▶RSA digital signature standard [73]), one requires that the hash function is not multiplicative, i.e., it should be hard to find inputs x, x', x'' such that $h(x) \cdot h(x') = h(x'')$ for some group operation "·". See section ▶Attacks Dependent . . . Scheme for an example and [3] for a more extensive discussion on these aspects.

If one digitally signs a hash value of a message rather than the message itself, or if one uses a hash function to commit to a value, a CRHF hash function is necessary. For other applications of hash functions, such as protecting passphrases, preimage resistance is sufficient.

Iterated Hash Functions: Most practical hash functions are based on a compression function with fixed size input; they process every message block in a similar way. Lai and Massey call this an *iterated* hash function [50]. The input is first padded such that the length of the input is a multiple of the block length. Next, it is divided into t blocks X_1 through X_t. The hash result is then computed as follows:

$$\begin{aligned} H_0 &= IV \\ H_i &= f(X_i, H_{i-1}), \quad i = 1, 2, \dots t \\ h(X) &= g(H_t). \end{aligned}$$

Here IV is the abbreviation of *Initial Value*, H_i is called the *chaining variable*, the function f is called the *compression* function or *round* function, and the function g is called the *output transformation*. Most constructions used for g is the identity function. Two elements in this definition have an important influence on the security of a hash function: the choice of the padding rule and the choice of the IV. It is essential that the padding rule is unambiguous (i.e., there do not exist two messages that can be padded to the same padded message); at the end one should append the length of the message; and the IV should be defined as part of the description of the hash function (this is called MD-strengthening after Merkle [53] and Damgård [21]).

A natural question can now be formulated: Which properties should be imposed on f to guarantee that h satisfies certain properties? Two partial answers have been found. The first result is by Lai and Massey [50] and gives necessary and sufficient conditions for f in order to obtain an *ideally secure* hash function h.

Theorem 1 **(Lai–Massey)** *Assume that the padding contains the length of the input string, and that the message X (without padding) contains at least 2 blocks. Then finding a second preimage for h with a fixed IV requires* 2^n *operations if and only if finding a second preimage for f with arbitrarily chosen* H_{i-1} *requires* 2^n *operations.*

The fact that the condition is necessary follows from the following argument: If one can find a second preimage for f in 2^s operations (with $s < n$), one can find a second preimage for h in $2^{(n+s)/2+1}$ operations with a ▶meet-in-the-middle attack (cf. section ▶Meet-in-the-Middle Attack).

A second result by Damgård [21] and independently by Merkle [53] states that for h to be a CRHF, it is sufficient that f is a collision resistant function.

Theorem 2 **(Damgård–Merkle)** *Let f be a collision resistant function mapping l to n bits (with* $l - n > 1$*). If an unambiguous padding rule is used, the following construction yields a CRHF:*

$$\begin{aligned} H_1 &= f(0^{n+1} \| X_1) \\ H_i &= f(H_{i-1} X_i), \quad \textit{for } i = 2, 3, \dots, t. \end{aligned}$$

The construction can be improved slightly, and extended to the case where $l = n + 1$, at the cost of an additional assumption on f (see [21] for details and Gibson's comment [37]). This variant, which is used in practice, avoids the prefixing of a zero or one bit. This construction can also extended to a tree, which allows for increased parallelism [21, 60].

Methods of Attack on Hash Functions: A distinction is made between attacks that only depend on the size n in bits of the hash result attacks that depend on the black-box properties of the compression function and cryptanalytic attacks that exploit the detailed structure of the compression function.

The security of hash functions is heuristic; only in a few slow constructions, it can be reduced to a number theoretic problem. Therefore, it is recommended to be conservative in selecting a hash function: One should not use hash functions for which one can find “near” collisions or (second) preimages, or hash functions for which one can only find “randomly looking” collisions or (second) preimages. Such a property may not be a problem for most applications, but one can expect that attacks can be refined to create collisions or (second) preimages that satisfy additional constraints. It is also important to note that many attacks only impose constraints on the last one or two blocks of the input.

Black-Box Attacks on Hash Functions

These attacks depend only on the size n in bits of the hash result; they are independent of the specific details of the algorithm. It is assumed that the hash function behaves as a random function: If this is not the case this class of attacks will typically be more effective.

For attacks that require a limited memory size and not too many memory accesses, 2^{70} operations is considered to be on the edge of feasibility (in 2004). In view of the fact that the speed of computers is multiplied by 4 every 3 years (this is one of the formulations of Moore’s law), 2^{80} operations is sufficient for the next 5–10 years, but it will be only marginally secure within 15 years. For applications that require protection for 20 years, one should try to design the scheme such that an attack requires at least 2^{90} operations. For a more detailed discussion of the cost of brute force attacks, refer the entry on ►Exhaustive Key Search.

Random (Second) Preimage Attack. One selects a random message and hopes that a given hash result will be obtained. If the hash function has the required “random” behavior, the success probability equals $1/2^n$. This attack can be carried out off-line and in parallel. If t hash results can be attacked simultaneously, the work factor is divided by a factor t, but the storage requirement becomes t n-bit blocks. For example, if $t = 2^{n/2}$, on average $2^{n/2}$ attempts are required to hit one of the values. This linear degradation of security can be mitigated by parameterizing the hash function and changing the parameter (or ►salt) for every instance [52] (►Preimage Resistance and ►Second Preimage Resistance).

Birthday Attack. The ►birthday paradox states that for a group of 23 people, the probability that at least two people have a common birthday exceeds 1/2. Intuitively one expects that the probability is much lower. However, the number of pairs of people in such a group equals $23 \times 22/2 = 253$. This can be exploited to find collisions for a hash function in the following way: One generates r_1 variations on a bogus message and r_2 variations on a genuine message. The expected number of collisions equals $r_1 \cdot r_2/n$. The probability of finding a bogus message and a genuine message that hash to the same result is given by $1 - \exp(-r_1 \cdot r_2/2^n)$, which is about 63% when $r = r_1 = r_2 = 2^{n/2}$. Finding the collision does not require r^2 operations: After sorting the data, which requires $O(r \log r)$ operations, the comparison is easy. This attack was first pointed out by Yuval [83].

One can substantially reduce the memory requirements (and also the memory accesses) for collision search by translating the problem to the detection of a cycle in an iterated mapping. This was first proposed by Quisquater and Delescaille [66]. Van Oorschot and Wiener propose an efficient parallel variant of this algorithm [80]; with a 10 million US$ machine, collisions for ►MD5 (with $n = 128$) can be found in 21 days in 1994, which corresponds to 5 h in 2004. In order to make a collision search infeasible, n should be at least 160 bits; security for 15–20 years or more requires at least 180 bits.

Black-Box Attacks on the Compression Function

This class of attacks depends on some high-level properties of the compression function f. They are also known as chaining attacks, since they exploit the way in which multiple compression functions are combined.

Meet-in-the-Middle Attack. This attack applies to hash functions for which the compression function f is easy to invert (see also Theorem 1). It is a variant of the birthday attack that allows to find a (second) preimage in time $2^{n/2}$ rather than 2^n. The opponent generates r_1 variations on the first part of a bogus message and r_2 variations on the last part. Starting from the initial value and going backward from the hash result, the probability for a matching intermediate variable is given by $1 - \exp(-r_1 \cdot r_2/2^n)$. The only restriction that applies to the meeting point is that it cannot be the first or last value of the chaining variable. The memory cost can be made negligible using a cycle finding algorithm [67]. For more details, refer the entry ►Meet-in-the-Middle Attack.

A generalized meet-in-the-middle attack has been proposed by Coppersmith [15] and Girault et al. [39] to break p-fold iterated schemes, i.e., weak schemes with more than one pass over the message as proposed by Davies and Price [22].

Correcting-Block Attack. This attack consists of substituting all blocks of the message except for one or more blocks. This attack often applies to the last block and is then called a correcting-last-block attack, but it can also apply to the first block or to some blocks in the middle. For a collision attack, one chooses two arbitrary messages X and X' with $X' \neq X$; subsequently one searches for one or more correcting blocks denoted with Y and Y', such that $h(X' \parallel Y') = h(X \parallel Y)$. A similar approach is taken for a (second) preimage attack. Hash functions based on algebraic structures are particularly vulnerable to this attack. Refer ►Correcting-Block Attack for more details.

Fixed Point Attack. A fixed point for a compression f is a pair (H_{i-1}, X_i) that satisfies $f(X_i, H_{i-1}) = H_{i-1}$. If the chaining variable is equal to H_{i-1}, one can insert an arbitrary number of blocks equal to X_i without modifying the hash result. Producing collisions or a second preimage with this attack is only possible if the chaining variable can be made equal to H_{i-1}: This is the case if IV can be chosen equal to a specific value, or if a large number of fixed points can be constructed (e.g., if one can find an X_i for a significant fraction of H_{i-1}'s). This attack can be made more difficult by appending a block count or bit count and by fixing IV (MD-strengthening, refer section Iterated Hash Functions).

Attacks Dependent on the Internal Details of the Compression Function

Most cryptanalytical techniques that have been applied to ►block ciphers have a counterpart for hash functions. As an example, ►differential cryptanalysis has been shown to be a very effective attack tool on hash functions [9]. Differential attacks of hash functions based on block ciphers have been studied in [62, 69]. Special cryptanalytic techniques have been invented by Dobbertin to cryptanalyze ►MD4, ►MD5, and the ►RIPEMD family [26, 27]. For hash functions based on block ciphers, weaknesses of block ciphers that may not be a problem for confidentiality protection can be problematic for hashing applications. For example, one needs to take into account the complementation property and fixed points [58] of ►DES, as well as the existence of ►weak keys (refer also the weak hash keys defined by Knudsen [47]).

Attacks Dependent on the Interaction with the Signature Scheme

Signature schemes can be made more efficient and secure by compressing the information to be signed with a hash function and to sign the hash result. Even if the hash function is collision resistant, it might be possible to break the resulting signature scheme. This attack is then the consequence of an interaction between both schemes. In the known examples of such an interaction, both the hash function and the signature scheme have some multiplicative structure (Coppersmith's attack on X.509 Annex D [16], ►Correcting Block Attack).

A more subtle point is that problems can arise if the hash function and the digital signature scheme are not coupled. For example, given $h(X)$, with h a strong CRHF, one could try to find a value X' such that $h'(X') = h(X)$, where h' is a weak hash function, and then claim that the signer has signed X' with h' instead of X with h. This problem can be addressed to some extent by signing together with the hash result a unique hash identifier (e.g., as defined in [42]). However, Kaliski has pointed out that this approach has some limitations [46]. A simpler approach is to allow only one hash function for a given signature scheme (DSA [34] and ►SHA-1 [32]).

An Overview of Practical Hash Functions: This section presents three types of hash functions: custom designed hash functions, hash functions based on a block cipher, and hash functions based on algebraic structures (modular arithmetic, knapsack, and lattice problems). It is important to be aware of the fact that many proposals have been broken; due to space limitations, it is not possible to include all proposals found in the literature and the attacks on them. For a more detailed discussion, the reader is referred to [65].

Custom Designed Hash Functions

This section discusses a selection of custom designed hash functions, i.e., algorithms that have been designed for hashing operations. Most of these algorithms use the Davies–Meyer approach (cf. section ►Hash Functions ... Cipher): The compression function is a block cipher, keyed by the text input X_i; the plaintext is the value H_{i-1}, which is also added to the ciphertext (feedforward).

MD2 [45] is a hash function with a 128-bit result that was published by Rivest in 1990. The algorithm is software oriented, but due to the byte structure it is not very fast on 32-bit machines. It inserts at the end a checksum byte (a weak hash function) of all the inputs. Rogier and Chauvaud have found collisions for a variant of MD2, that omits the checksum byte at the end [75].

A much faster algorithm by the same designer is ►MD4 [71]; it also dates back to 1990 and has a 128-bit result. It is defined for messages of shorter than 2^{64} bits. An important contribution of MD4 is that it has introduced innovative design principles; it was the first published cryptographic algorithm that made optimal use of logic operations and integer arithmetic on 32-bit processors. The compression function hashes a 128-bit chaining variable and a 512-bit message block to a 128-bit chaining variable. The algorithms derived from it (with improved strength) are often called the *MDx-family*. This family contains the most popular hash functions used today. Dobbertin has found collisions for MD4; his attack combines algebraic techniques and optimization techniques such as genetic algorithms [26, 27]. It can be extended to result in "meaningful" collisions: The complete message (except for a few dozen bytes) is under complete control of the attacker. His attack also applies to the compression function of *extended MD4* [71], which consists of two loosely coupled instances of MD4. Later Dobbertin et al. showed that a reduced version of MD4 (2 rounds out of 3) is not preimage resistant [29].

Following early attacks on MD4 by den Boer and Bosselaers [23] and by Merkle, Rivest quickly proposed a strengthened version, namely ►MD5 [72]. It was however shown by den Boer and Bosselaers [24] that the compression function of MD5 is not collision resistant (but their collisions are of the special form $f(H_{i-1}, X_i) = f(H'_{i-1}, X_i)$, which implies they have no direct impact on applications). Dobbertin has extended his attack on MD4 to yield collisions for the compression function of MD5, i.e., $f(H_{i-1}, X_i) = f(H_{i-1}, X'_i)$, where he has some control over H_{i-1} [28]. It is believed that it is feasible to extend this attack to collisions for MD5 itself (i.e., to take into account the IV).

HAVAL was proposed by Zheng et al. at Auscrypt'92 [86]; it consists of several variants (outputs length between 128 and 256 bits and 3, 4, or 5 rounds). The 3-round version was broken by Van Rompay et al. [81] for all output lengths.

NIST has published a series of variants on MD4 as FIPS standards under the name Secure Hash Algorithm family or ►SHA family. The first Secure Hash Algorithm was published by NIST [31] in 1993 (it is now referred to as SHA-0). The size of the hash result is 160 bits. In 1995, NIST discovered a certificational weakness in SHA-0, which resulted in a new release of the standard published under the name SHA-1 [32]. In 1998, Chabaud and Joux have published an attack that finds collisions for SHA-0 in 2^{61} operations [13]; it is probably similar to the (classified) attack developed earlier that prompted the upgrade to SHA-1. In 2002, three new hash functions have been published with longer hash results: SHA-256, SHA-384, and SHA-512 [33]. In December 2003, SHA-224 has been added in a change notice to [33]. SHA-256 and SHA-224 have eight 32-bit chaining variables, and their compression function takes message blocks of 512 bits and chaining variables of 256 bits. SHA-384 and SHA-512 operate on 64-bit words; their compression function processes messages in blocks of 1024 bits and chaining variables of 512 bits. They are defined for messages shorter than 2^{128} bits.

Yet another improved version of MD4, called RIPEMD, was developed in the framework of the EEC-RACE project RIPE [70]. RIPEMD has two independent paths with strong interaction at the end of the compression function. It resulted later in the ►RIPEMD family. Due to partial attacks by Dobbertin [26], RIPEMD was upgraded by Dobbertin et al. to RIPEMD-128 and RIPEMD-160, which have a 128-bit and a 160-bit result, respectively [30]. Variants with a 256- and 320-bit result have been introduced as well.

Whirlpool is a design by Barreto and V. Rijmen [6]; it consists of a 512-bit block cipher with a 512-bit key in the Miyaguchi–Preneel mode (refer section ►Hash Functions . . . Cipher); it offers a result of 512 bits. The design principles of Whirlpool are closely related to those of ►Rijndael/AES.

Together with SHA-256, SHA-384, and SHA-512, Whirlpool has been recommended by the ►NESSIE project. The ISO standard on design hash functions (ISO/IEC 10118-3) contains RIPEMD-128, RIPEMD-160, SHA-1, SHA-256, SHA-384, SHA-512, and Whirlpool [42]. Other custom designed hash functions include FFT-Hash III [76], N-hash [57], Snefru [54], Subhash [18], and Tiger [4].

Hash Functions Based on a Block Cipher

Hash functions based on a block cipher have been popular as this construction limits the number of cryptographic algorithms which need to be evaluated and implemented. The historic importance of ►DES, which was the first standard commercial cryptographic primitive that was widely available, also plays a role. The main disadvantage of this approach is that custom designed hash functions are likely to be more efficient. This is particularly true because hash functions based on block ciphers require a key change after every encryption.

The encryption operation E will be written as $Y = E_K(X)$. Here X denotes the plaintext, Y the ciphertext, and K the key. The size of the plaintext and ciphertext or the block length (in bits) will be denoted with r, while the key size (in bits) will be denoted with k. For ►DES, $r = 64$ and $k = 56$. The *hash rate* of a hash function based on a block cipher is defined as the number of r-bit input blocks that can be processed with a single encryption.

The discussion in this section will be limited to the case $k \approx r$. For $k \approx r$, a distinction will be made between the cases $n = r$, $n = 2r$, and $n > 2r$. Next the case $k \approx 2r$ will be discussed. A more extensive treatment can be found in [65].

Size of Hash Result Equal to the Block Length. All known schemes of this type have rate 1. The first secure construction for such a hash function was the 1985 scheme by Matyas et al. [51]:

$$H_i = E^{\oplus}_{s(H_{i-1})}(X_i).$$

Here $s()$ is a mapping from the ciphertext space to the key space and $E^{\oplus}_K(X)$ denotes $E_K(X) \oplus X$. This scheme has been included in ISO/IEC 10118–2 [42]. The dual of this scheme is the Davies–Meyer scheme:

$$H_i = E^{\oplus}_{X_i}(H_{i-1}). \quad (1)$$

It has the advantage that it extends more easily to block ciphers for which key size and block size are different. A variant on these schemes was proposed in 1989 independently by Miyaguchi et al. [43, 57] and Preneel et al. [63] (it is known as the Miyaguchi–Preneel scheme):

$$H_i = E^{\oplus}_{s(H_{i-1})}(X_i) \oplus H_{i-1}.$$

In 1993, Preneel et al. identify 12 secure variants [64]; Black et al. [10] offer a concrete security proof for these schemes in the black-box cipher model (after earlier work in [82]): This implies that in this model, finding a collision requires $\approx 2^{r/2}$ encryptions and finding a (second) preimage takes $\approx 2^r$ encryptions. This shows that these hash functions can only be collision resistant if the block length is 160 bits or more. Most block ciphers have a smaller block length, which motivates the constructions in the next section.

Size of Hash Result Equal to Twice the Block Length. The goal of *double block length* hash functions is to achieve a higher security level against collision attacks. Ideally a collision attack on such a hash function should require 2^r encryptions, and a (second) preimage attack 2^{2r} encryptions.

The few proposals that survive till today have rate less than 1. Two important examples are ▸MDC-2 and ▸MDC-4 with hash rate 1/2 and 1/4, respectively. They have been designed by Brachtl et al. [12], and are also known as the Meyer–Schilling hash functions after the authors of the first paper described these schemes [55].

$$\begin{aligned} T^1_i &= E^{\oplus}_{u(H^1_{i-1})}(X_i) = LT^1_i \| RT^1_i \\ T^2_i &= E^{\oplus}_{v(H^2_{i-1})}(X_i) = LT^2_i \| RT^2_i \\ H^1_i &= LT^1_i \| RT^2_i \\ H^2_i &= LT^2_i \| RT^1_i \,. \end{aligned}$$

The variables H^1_0 and H^2_0 are initialized with the values IV_1 and IV_2 respectively, and the hash result is equal to the concatenation of H^1_t and H^2_t. The functions u, v map the ciphertext space to the key space and need to satisfy the condition $u(IV^1) \neq v(IV^2)$. For $k = r$, the best known preimage and collision attacks on MDC-2 require $2^{3r/2}$ and 2^r operations, respectively [50]. A collision attack on MDC-2 based on the ▸Data Encryption Standard (DES) ($r = 64$, $k = 56$) requires at most 2^{55} encryptions. Note that the compression function of MDC-2 is rather weak, hence Theorems 1 and 2 cannot be applied: Preimage and collision attacks on the compression function require at most 2^r and $2^{r/2}$ operations. MDC-2 has been included in ISO/IEC 10118-2 [42].

One iteration of ▸MDC-4 consists of the concatenation of two MDC-2 steps, where the plaintexts in the second step are equal to $H2_{i-1}$ and $H1_{i-1}$. The rate of MDC-4 is equal to 1/4. For $k = r$, the best known preimage attack for MDC-4 requires $2^{7r/4}$ operations. This shows that MDC-4 is probably more secure than MDC-2 against preimage attacks. However, finding a collision for MDC-4 itself requires only 2^{r+2} encryptions, while finding a collision for its compression function requires $2^{3r/4}$ encryptions [49, 62]. The best known (second) preimage attack on the compression function of MDC-4 requires $2^{3r/2}$ encryptions.

A series of proposals attempted to achieve rate 1 with constructions of the following form:

$$\begin{aligned} H^1_i &= E_{A^1_i}\left(B^1_i\right) \oplus C^1_i \\ H^2_i &= E_{A^2_i}\left(B^2_i\right) \oplus C^2_i, \end{aligned}$$

where A^1_i, B^1_i, and C^1_i are binary linear combinations of H^1_{i-1}, H^2_{i-1}, X^1_i, and X^2_i and where A^2_i, B^2_i, and C^2_i are binary linear combinations of H^1_{i-1}, H^2_{i-1}, X^1_i, X^2_i, and H^1_i. The hash result is equal to the concatenation of H^1_t and H^2_t. However, Knudsen et al. showed that for all hash functions in this class, a preimage attack requires at most 2^r operations, and a collision attack requires at most $2^{3r/4}$ operations (for most schemes this can be reduced to $2^{r/2}$) [48].

Merkle describes an interesting class of proposals in [53], for which he proves in the black-box cipher model that the compression function is collision resistant. The most efficient scheme has rate 1/4 and offers a security level of 2^{56} encryptions when used with DES. Other results on output length doubling have been obtained by Aiello and Venkatesan [1].

Size of Hash Result Larger than Twice the Block Length. Knudsen and Preneel propose a collision resistant compression function, but with parallel encryptions only [49]. They show how a class of efficient constructions for hash functions can be obtained based on non-binary error-correcting codes. Their schemes can achieve a provable security level against collisions equal to 2^r, $2^{3r/2}$ (or more) and this with rates larger than 1/2 (based on a rather strong assumption). The internal memory of the scheme is larger than two or three blocks, which implies that an output transformation is required. Two of their schemes have been included in ISO/IEC 10118-2 [42].

Size of the Key Equal to Twice the Block Length. In this case, making efficient constructions is a little easier. A scheme in this class was proposed by Merkle [52], who observed that a block cipher with key length larger than block length is a natural compression function:

$$H_i = E_{H_{i-1} \| X_i}(C),$$

with C a constant string. Another construction can be found in [50].

Two double length hash functions with rate 1/2 have been proposed by Lai and Massey [50]; they form extensions of the Davies–Meyer scheme. One scheme is called *Tandem Davies–Meyer*, and has the following description:

$$H_i^1 = E^{\oplus}_{H_{i-1}^2 \| X_i}\left(H_{i-1}^1\right)$$
$$H_i^2 = E^{\oplus}_{X_i \| (H_i^1 \oplus H_{i-1}^1)}\left(H_{i-1}^2\right)$$

The second scheme is called *Abreast Davies–Meyer*:

$$H_i^1 = E^{\oplus}_{H_{i-1}^2 \| X_i}\left(H_{i-1}^1\right)$$
$$H_i^2 = E^{\oplus}_{X_i \| H_{i-1}^1}\left(\bar{H}_{i-1}^2\right).$$

Here $\bar{H}$ denotes the bitwise complement of H. The best known attacks for a preimage and a collision require 2^{2r} and 2^r encryptions, respectively. Faster schemes in this class have been developed in [49].

Hash Functions Based on Algebraic Structures

It should be pointed out that several of these hash functions are vulnerable to the insertion of trapdoors, which allow the person who chooses the design parameters to construct collisions. Therefore, one needs to be careful with the generation of the instance. For an ▸RSA public key encryption modulus, one could use a distributed generation as developed by Boneh and Franklin [11] and Frankel et al. [35].

Hash Functions Based on Modular Arithmetic. For several schemes, there exists a security reduction to a number theoretic problem that is believed to be difficult. However, they are very slow: Typically, they hash $\log_2 \log_2 N$ bits per modular squaring (or even per modular exponentiation). Damgård provides two hash functions for which finding a collision is provably equivalent to factoring an RSA modulus [19]. Gibson proposes a construction based on the discrete logarithm problem modulo a composite [38]. A third approach by Bellare et al. [7] uses the discrete logarithm problem in a group of prime order p denoted with G_p. Every nontrivial element of G_p is a generator. The hash function uses t random elements α_i from G_p $(\alpha_i \neq 1)$. The hash result is then computed as

$$H_{t+1} = \prod_{i=1}^{t} \alpha_i^{\tilde{X}_i}.$$

Here $\tilde{X}_i$ is obtained by considering the string X_i as the binary expansion of a number and prepending 1 to it. This avoids trivial collisions when X_i consists of all zeros.

There exists a large number of ad hoc schemes for which there is no security reduction; many of these have been broken. The most efficient schemes are based on modular squaring. The best schemes seem to be of the form:

$$H_i = \left((X_i \oplus H_{i-1})^2 \bmod N\right) \oplus H_{i-1}.$$

In order to preclude a ▸correcting block attack, it is essential to add redundancy to the message blocks X_i and to perform some additional operations. Two constructions, ▸MASH-1 and ▸MASH-2 (for Modular Arithmetic Secure Hash) have been standardized in ISO/IEC 10118-4 [42]. MASH-1 has the following compression function:

$$H_i = \left(\left((X_i \oplus H_{i-1}) \vee A\right)^2 (\bmod N)\right) \oplus H_{i-1},$$

Here A = `0xF00...00`, the four most significant bits in every byte of X_i are set to `1111`, and the output of the squaring operation is chopped to n bits. A complex output transformation is added, which consists of a number of applications of the compression function; its goal is to destroy all the remaining mathematical structure. The final result is at most $n/2$ bits. The best known preimage and collision attacks on MASH-1 require $2^{n/2}$ and $2^{n/4}$ operations [17]; they are thus not better than brute force attacks. MASH-2 is a variant of MASH-1 which uses exponent 2^8+1 [42]. This provides for an additional security margin.

Hash Functions Based on Knapsack and Lattice Problems. The ▸knapsack problem (▸Knapsack Cryptographic Schemes) of dimensions n and $\ell(n)$ can be defined as follows: given a set of n l-bit integers $\{a_1, a_2, \ldots, a_n\}$, and an integer S, find a vector X with components x_i equal to 0 or 1 such that

$$\sum_{i=1}^{n} a_i \cdot x_i = S \bmod 2^{\ell(n)}.$$

For application to hashing, one needs $n > \ell(n)$; knapsack problems become more difficult when $n \approx \ell(n)$; however, the performance of the hash function decreases with the value $n - \ell(n)$. The best known attacks are those based on ▸lattice reduction (LLL) [44] and an algebraic technique which becomes more effective if $n \gg \ell(n)$ [61]. It remains an open problem whether for practical instances a random knapsack is sufficiently hard.

Ajtai introduced a function that is one-way (or preimage resistant) if the problem of approximating the shortest vector in a lattice to polynomial factors is hard [2]. Goldreich et al. have proved that the function is in fact collision resistant [40]. Micciancio has proposed a CRHF for which the security is based on the worst-case hardness of approximating the covering radius of a lattice [56].

Several multiplicative knapsacks have also been proposed, such as the schemes by Zémor [84] and Tillich and Zémor [79]. Their security is based on the hardness of finding short factorizations in certain groups. In some cases, one can even prove a lower bound on the Hamming distance between colliding messages. Attacks on these proposals (for certain parameters) can be found in [14, 36].

Incremental Hash Functions. A hash function (or any cryptographic primitive) is called *incremental* if it has the following property: If the hash function has been evaluated for an input x, and a small modification is made to x, resulting in x', then one can update $h(x)$ in time proportional to the amount of modification between x and x', rather than having to recompute $h(x')$ from scratch. If a function is incremental, it is automatically parallelizable as well.

This concept was first introduced by Bellare et al. [7]. They also made a first proposal based on exponentiation in a group of prime order. Improved constructions were proposed by Bellare and Micciancio [8].

Recommended Reading

1. Aiello W, Venkatesan R (1996) Foiling birthday attacks in length-doubling transformations. Benes: a non-reversible alternative to Feistel. In: Maurer U (ed) Advances in cryptology – EUROCRYPT'96. Lecture notes in computer science, vol 1070. Springer, Berlin, pp 307–320
2. Ajtai M (1996) Generating hard instances of lattice problems. In: Proceedings of 28th ACM symposium on the theory of computing, Philadelphia, pp 99–108
3. Anderson R (1995) The classification of hash functions. In: Farrell PG (ed) Codes and cyphers: cryptography and coding IV. Institute of Mathematics & Its Applications (IMA), Southend-on-Sea, pp 83–93
4. Anderson R, Biham E (1996) Tiger: a new fast hash function. In: Gollmann D (ed) Fast software encryption. Lecture notes in computer science, vol 1039. Springer, Berlin, pp 89–97
5. Anderson R, Biham E (1996) Two practical and provably secure block ciphers: BEAR and LION. In: Gollmann D (ed) Fast software encryption. Lecture notes in computer science, vol 1039. Springer, Berlin, pp 113–120
6. Barreto PSLM, Rijmen V (2000) The Whirlpool hashing function. NESSIE submission
7. Bellare M, Goldreich O, Goldwasser S (1994) Incremental cryptography: the case of hashing and signing. In: Desmedt Y (ed) Advances in cryptology – CRYPTO'94. Lecture notes in computer science, vol 839. Springer, Berlin, pp 216–233
8. Bellare M, Micciancio D (1997) A new paradigm for collision-free hashing: incrementality at reduced cost. In: Fumy W (ed) Advances in cryptology – EUROCRYPT'97. Lecture notes in computer science, vol 1233. Springer, Berlin, pp 163–192
9. Biham E, Shamir A (1993) Differential cryptanalysis of the data encryption standard. Springer, Berlin
10. Black J, Rogaway P, Shrimpton T (2002) Black-box analysis of the block-cipher based hash function constructions from PGV. In: Yung M (ed) Advances in cryptology – CRYPTO 2002. Lecture notes in computer science, vol 2442. Springer, Berlin, pp 320–355
11. Boneh D, Franklin M (1997) Efficient generation of shared RSA keys. In: Kaliski B (ed) Advances in cryptology – CRYPTO'97. Lecture notes in computer science, vol 1294. Springer, Berlin, pp 425–439
12. Brachtl BO, Coppersmith D, Hyden MM, Matyas SM, Meyer CH, Oseas J, Pilpel S, Schilling M (1990) Data authentication using modification detection codes based on a public one way encryption function, U.S. Patent Number 4,908,861, 13 Mar 1990
13. Chabaud F, Joux A (1998) Differential collisions: an explanation for SHA-1. In: Krawczyk H (ed) Advances in cryptology – CRYPTO'98. Lecture notes in computer science, vol 1462. Springer, Berlin, pp 56–71
14. Charnes C, Pieprzyk J (1995) Attacking the SL_2 hashing scheme. In: Pieprzyk J, Safavi-Naini R (eds) Advances in cryptography – ASIACRYPT'94. Lecture notes in computer science, vol 917. Springer, Berlin, pp 322–330
15. Coppersmith D (1985) Another birthday attack. In: Williams HC (ed) Advances in cryptology – CRYPTO'85. Lecture notes in computer science, vol 218. Springer, Berlin, pp 14–17
16. Coppersmith D (1989) Analysis of ISO/CCITT document X.509 Annex D. IBM T.J. Watson Center, Yorktown Heights, Internal Memo, 11 June 1989 (also ISO/IEC JTC1/SC20/WG2/N160)
17. Coppersmith D, Preneel B (1995) Comments on MASH-1 and MASH-2. ISO/IEC JTC1/SC27/N1055. Accessed 21 Feb 1995
18. Daemen J (1995) Cipher and hash function design. Strategies based on linear and differential cryptanalysis. Doctoral dissertation, Katholieke Universiteit Leuven
19. Damgård IB (1988) Collision free hash functions and public key signature schemes. In: Chaum D, Price WL (eds) Advances in cryptology – EUROCRYPT'87. Lecture notes in computer science, vol 304. Springer, Berlin, pp 203–216
20. Damgård IB (1988) The application of claw free functions in cryptography. Ph.D. thesis, Aarhus University, Mathematical Institute
21. Damgård IB (1990) A design principle for hash functions. In: Brassard G (ed) Advances in cryptology – CRYPTO'89. Lecture notes in computer science, vol 435. Springer, Berlin, pp 416–427
22. Davies D, Price WL (1980) The application of digital signatures based on public key cryptosystems. NPL Report, DNACS 39/80, Dec 1980
23. den Boer B, Bosselaers A (1992) An attack on the last two rounds of MD4. In: Feigenbaum J (ed) Advances in cryptology – CRYPTO'91. Lecture notes in computer science, vol 576. Springer, Berlin, pp 194–203
24. den Boer B, Bosselaers A (1994) Collisions for the compression function of MD5. In: Helleseth T (ed) Advances in cryptology – EUROCRYPT'93. Lecture notes in computer science, vol 765. Springer, Berlin, pp 293–304

25. Diffie W, Hellman ME (1976) New directions in cryptography. IEEE T Info Theory IT-22(6):644–654
26. Dobbertin H (1997) RIPEMD with two-round compress function is not collision-free. J Cryptol 10(1):51–69
27. Dobbertin H (1998) Cryptanalysis of MD4. J Cryptol 11(4):253–271. See also Gollmann D (ed) (1996) Fast software encryption. Lecture notes in computer science, vol 1039. Springer, Berlin, pp 53–69
28. Dobbertin H (1996) The status of MD5 after a recent attack. CryptoBytes 2(2):1–6
29. Dobbertin H (1998) The first two rounds of MD4 are not one-way. In: Vaudenay S (ed) Fast software encryption. Lecture notes in computer science, vol 1372. Springer, Berlin, pp 284–292
30. Dobbertin H, Bosselaers A, Preneel B (1996) RIPEMD-160: a strengthened version of RIPEMD. In: Gollmann D (ed) Fast software encryption. Lecture notes in computer science, vol 1039. Springer, Berlin, pp 71–82. See also http://www.esat.kuleuven.ac.be/~bosselae/ripemd160
31. FIPS 180 (1993) Secure hash standard. Federal information processing standard (FIPS), Publication 180. National Institute of Standards and Technology, US Department of Commerce, Washington, DC, 11 May 1993
32. FIPS 180-1 (1995) Secure hash standard. Federal information processing standard (FIPS), Publication 180–1. National Institute of Standards and Technology, US Department of Commerce, Washington, DC, 17 Apr 1995
33. FIPS 180-2 (2003) Secure hash standard. Federal information processing standard (FIPS), Publication 180–2. National Institute of Standards and Technology, US Department of Commerce, Washington, DC, 26 Aug 2002 (Change notice 1 published on 1 Dec)
34. FIPS 186 (1994) Digital signature standard. Federal information processing standard (FIPS), Publication 186. National Institute of Standards and Technology, US Department of Commerce, Washington, DC, 19 May 1994
35. Frankel Y, MacKenzie PD, Yung M (1998) Robust efficient distributed RSA-key generation. In: Proceedings of 30th ACM symposium on the theory of computing, Dallas, pp 663–672
36. Geiselmann W (1995) A note on the hash function of Tillich and Zémor. In: Boyd C (ed) Cryptography and Coding, fifth IMA Conference. Springer, Berlin, pp 257–263
37. Gibson JK (1990) Some comments on Damgård's hashing principle. Electron Lett 26(15):1178–1179
38. Gibson JK (1991) Discrete logarithm hash function that is collision free and one way. IEEE Proc 138(6):407–410
39. Girault M, Cohen R, Campana M (1988) A generalized birthday attack. In: Günther CG (ed) Advances in cryptology – EUROCRYPT'88. Lecture notes in computer science, vol 330. Springer, Berlin, pp 129–156
40. Goldreich O, Goldwasser S, Halevi S (1996) Collision-free hashing from lattice problems. Theory of Cryptography Library. http://philby.ucsd.edu/cryptolib.html
41. Handschuh H, Knudsen LR, Robshaw MJB (2001) Analysis of SHA-1 in encryption mode. In: Naccache D (ed) Topics in cryptology – CT-RSA 2001. Lecture notes in computer science, vol 2020. Springer, Berlin, pp 70–83
42. ISO/IEC 10118 (2000) Information technology – security techniques – hash-functions, Part 1: general, Part 2: hash-functions using an n-bit block cipher algorithm, (2003) Part 3: dedicated hash-functions, (1998) Part 4: hash-functions using modular arithmetic
43. ISO-IEC/JTC1/SC27/WG2 N98 (1991) Hash functions using a pseudo random algorithm. Japanese contribution
44. Joux A, Granboulan L (1995) A practical attack against knapsack based hash functions. In: De Santis A (ed) Advances in cryptology – EUROCRYPT'94. Lecture notes in computer science, vol 950. Springer, Berlin, pp 58–66
45. Kaliski Jr BS (1992) The MD2 message-digest algorithm. Request for comments (RFC) 1319, Internet Activities Board, Internet Privacy Task Force
46. Kaliski Jr BS (2002) On hash function firewalls in signature schemes. In: Preneel B (ed) Topics in cryptology – CT-RSA 2002. Lecture notes in computer science, vol 2271. Springer, Berlin, pp 1–16
47. Knudsen LR (1995) New potentially 'weak' keys for DES and LOKI. In: De Santis A (ed) Advances in cryptology – EUROCRYPT'94. Lecture notes in computer science, vol 950. Springer, Berlin, pp 419–424
48. Knudsen LR., Lai X, Preneel B (1998) Attacks on fast double block length hash functions. J Cryptol 11(1):59–72
49. Knudsen L, Preneel B (2002) Enhancing the security of hash functions using non-binary error correcting codes. IEEE T Info Theory 48(9):2524–2539
50. Lai X, Massey JL (1993) Hash functions based on block ciphers. In: Rueppel RA (ed) Advances in cryptology – EUROCRYPT'92. Lecture notes in computer science, vol 658. Springer, Berlin, pp 55–70
51. Matyas SM, Meyer CH, Oseas J (1985) Generating strong one-way functions with cryptographic algorithm. IBM Technol Discl Bull 27(10A):5658–5659
52. Merkle R (1979) Secrecy, authentication, and public key systems. UMI Research Press, Ann Arbor
53. Merkle R (1990) One way hash functions and DES. In: Brassard G (ed) Advances in cryptology – CRYPTO'89. Lecture notes in computer science, vol 435. Springer, Berlin, pp 428–446
54. Merkle R (1990) A fast software one-way hash function. J Cryptol 3(1):43–58
55. Meyer CH, Schilling M (1998) Secure program load with manipulation detection code. In: Proceedings of SECURICOM, Paris, pp 111–130
56. Micciancio D (2002) Improved cryptographic hash functions with worst-case/average case connection. In: Proceedings of 34th annual ACM symposium on theory of computing, Montréal, pp 609–618
57. Miyaguchi S, Iwata M, Ohta K (1989) New 128-bit hash function. In: Proceeding of fourth international joint workshop on computer communications, Tokyo, 13–15 July 1989, pp 279–288
58. Moore JH, Simmons GJ (1987) Cycle structure of the DES for keys having palindromic (or antipalindromic) sequences of round keys. IEEE T Softw Eng 13:262–273
59. Naor M, Yung M (1990) Universal one-way hash functions and their cryptographic applications. In: Proceedings of 21st ACM symposium on the theory of computing, Seattle, pp 387–394
60. Pal P, Sarkar P (2003) PARSHA-256 – a new parallelizable hash function and a multithreaded implementation. In: Johansson T (ed) Fast software encryption. Lecture notes in computer science, vol 2887. Springer, Berlin, pp 347–361
61. Patarin J (1995) Collisions and inversions for Damgård's whole hash function. In: Pieprzyk J, Safavi-Naini R (eds) Advances in cryptography – ASIACRYPT'94. Lecture notes in computer science, vol 917. Springer, Berlin, pp 307–321

62. Preneel B (1993) Analysis and design of cryptographic hash functions. Doctoral dissertation, Katholieke Universiteit Leuven
63. Preneel B, Govaerts R, Vandewalle J (1989) Cryptographically secure hash functions: an overview. ESAT Internal Report, K.U. Leuven
64. Preneel B, Govaerts R, Vandewalle J (1994) Hash functions based on block ciphers: a synthetic approach. In: Stinson D (ed) Advances in cryptology – CRYPTO'93. Lecture notes in computer science, vol 773. Springer, Berlin, pp 368–378
65. Preneel B (2004) Hash functions and MAC algorithms: state of the art. In: Preneel B (ed) Lecture notes in computer science. Springer, Berlin, in print
66. Quisquater J-J, Delescaille J-P (1990) How easy is collision search? Application to DES. In: Quisquater J-J, Vandewalle J (eds) Advances in cryptology – EUROCRYPT'89. Lecture notes in computer science, vol 434. Springer, Berlin, pp 429–434
67. Quisquater J-J, Delescaille J-P (1990) How easy is collision search. New results and applications to DES. In: Brassard G (ed) Advances in cryptology – CRYPTO'89. Lecture notes in computer science, vol 435. Springer, Berlin, pp 408–413
68. Rabin MO (1978) Digitalized signatures. In: Lipton R, DeMillo R (eds) Foundations of secure computation. Academic, New York, pp 155–166
69. Rijmen V, Preneel B (1995) Improved characteristics for differential cryptanalysis of hash functions based on block ciphers. In: Preneel B (ed) Fast software encryption. Lecture notes in computer science, vol 1008. Springer, Berlin, pp 242–248
70. RIPE (1995) Integrity primitives for secure information systems. In: Bosselaers A, Preneel B (eds) Final report of RACE integrity primitives evaluation (RIPE-RACE 1040). Lecture notes in computer science, vol 1007. Springer, Berlin
71. Rivest RL (1991) The MD4 message digest algorithm. In: Vanstone S (ed) Advances in cryptology – CRYPTO'90. Lecture notes in computer science, vol 537. Springer, Berlin, pp 303–311
72. Rivest RL (1992) The MD5 message-digest algorithm. Request for comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force
73. Rivest RL, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21:120–126
74. Rogaway P, Shrimpton T (2004) Cryptographic hash function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In: Meier W, Roy BK (eds) Fast software encryption. Lecture notes in computer science, vol 3017. Springer, Berlin, pp 371–388
75. Rogier N, Chauvaud P (1997) MD2 is not secure without the checksum byte. Design Code Cryptogr 12 (3), 245–251
76. Schnorr CP, Vaudenay S (1994) Parallel FFT-hashing. In: Anderson R (ed) Fast software encryption. Lecture notes in computer science, vol 809. Springer, Berlin, pp 149–156
77. Simon D (1998) Finding collisions on a oneway street: can secure hash functions be based on general assumptions? In: Nyberg K (ed) Advances in cryptology – EUROCRYPT'98. Lecture notes in computer science, vol 1403. Springer, Berlin, pp 334–345
78. Stinson D (2001) Some observations on the theory of cryptographic hash functions. Technical Report 2001/020, University of Waterloo
79. Tillich J-P, Zémor G (1994) Hashing with SL_2. In: Desmedt Y (ed) Advances in cryptology – CRYPTO'94. Lecture notes in computer science, vol 839. Springer, Berlin, pp 40–49
80. van Oorschot PC, Wiener M (1999) Parallel collision search with cryptanalytic applications. J Cryptol 12(1):1–28
81. Rompay V, Biryukov BA, Preneel B, Vandewalle J (2003) Cryptanalysis of 3-pass HAVAL. In: Lai CS (ed) Advances in cryptography – ASIACRYPT 2003. Lecture notes in computer science, vol 2894. Springer, Berlin, pp 228–245
82. Winternitz R (1984) A secure one-way hash function built from DES. In: Proceedings of the IEEE symposium on information security and privacy. IEEE Press, Los Alamitos, pp 88–90
83. Yuval G (1979) How to swindle Rabin. Cryptologia 3: 187–189
84. Zémor G (1994) Hash functions and Cayley graphs. Design Code Cryptogr 4(4):381–394
85. Zheng Y, Matsumoto T, Imai H (1990) Connections between several versions of one-way hash functions. Trans IEICE E E73(7):1092–1099
86. Zheng Y, Pieprzyk J, Seberry J (1993) HAVAL – a one-way hashing algorithm with variable length output. In: Seberry J, Zheng Y (eds) Advances in cryptology – AUSCRYPT'92. Lecture notes in computer science, vol 718. Springer, Berlin, pp 83–104

Hash-Based Message Authentication Code

►HMAC

Header Injections

►Header-Based Attacks

Header-Based Attacks

Serge Fenet
LIRIS UMR 5205, Université Claude Bernard Lyon 1, Villeurbanne cedex, France

Synonyms

Header injections

Definition

Header-based attacks are a form of computer offensive in which the attacker uses its ability to forge arbitrary header data to exploit a flaw in the target's software that will process this header. This flaw can reside in the code making the processing, but also, more dangerously, in the protocol describing this processing.

Although this kind of attack can be used with any level of the protocol stack, the term "Header Attack" is increasingly used to describe application-level attacks, for example, HTTP (Hypertext Transfer Protocol) header attacks.

Background

The sole principle of exchanging data on an unreliable medium with remote computers imposes to solve problems that naturally emerge: interacting with low level communication media, addressing remote computers, finding a route in the interconnection network, managing fragmentation and virtual connections, etc. All these requirements are managed by different protocol layers, structured in a stack, using the encapsulation principle: the N^{th} protocol layer uses the services of the $N - 1^{th}$ layer and provides services to the $N + 1^{th}$ layer (see Fig. 1).

In order to perform efficiently, each level of the stack has a direct communication link with its remote interlocutor. This mechanism is implemented by using a structured Protocol Data Units (PDU) that encompasses the original data that must be sent (the *Data* part, or Service Data Unit (SDI)), and the additional data required to implement the protocol (the *Header* part, or Protocol Control Information(PCI)). In the sender stack, the PDU of each level becomes the SDI of the level below, which adds it own PCI, creating a new PDU of its own level. This processing is reversed in the receiver stack.

Thus, while the payload is actually sent in the SDI, the additional data required by the communication between distant layers travels along within the PCI, as associated header.

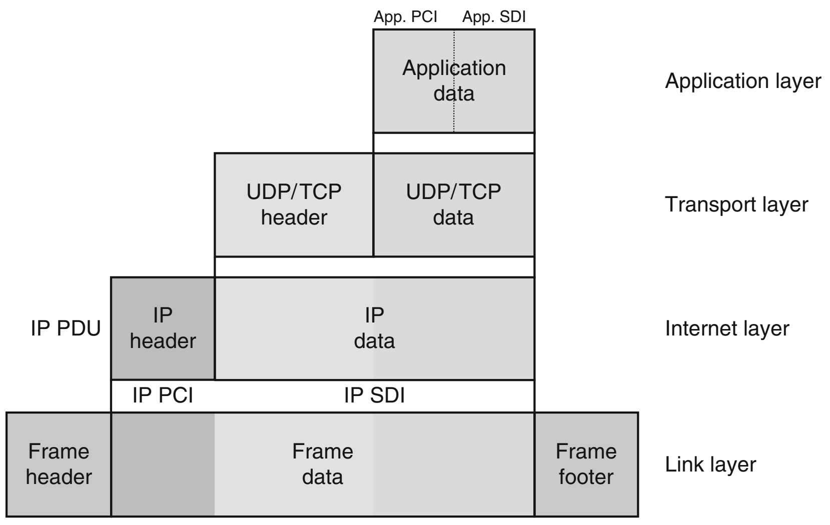
Header attacks are based on the ability to bypass the rules of SDI building programmed in the TCP/IP (Transmission Control Protocol/Internet Protocol) stack of the operating system [1], and to insert any chosen data in the newly created SDIs. This process is called "forging" a PDU.

Theory

The principle of header attacks is quite simple: Knowing how a given layer will handle the incoming PCI regarding the protocol it implements, an attacker will alter the legitimate data of this PCI, thus forging it, in order to induce a nonstandard behavior within the target system. This nonstandard behavior can be linked to an implementation flaw, for example, in the case of the Christmas Tree attack, or to a protocol flaw, like in the case of IP Spoofing or TCP connection theft. Of course, nothing can prevent an attacker to also forge the SDI part of the PDU, so that any data may easily be sent on any given link.

Header Attacks Below the Application Layer

The lower parts of the TCP/IP protocol stack (link, Internet, and transport layers) are the implementation framework for computer network protocols described as soon as the 1970s. Few changes have been made to this part of the stack since these years and, indeed, all computers using TCP/Ipv4 (Internet Protocol version 4) nowadays rely doubtlessly on very efficient, yet old protocols. The concerns of these older days, namely, end to end and robustness principles, ignored the security aspects that have become most important nowadays. Thus, properties like confidentiality, authenticity, or non-repudiation have to be implemented by additional protocols like IPSec (Internet Protocol Security) or SSL (Secure Sockets Layer)



Header-Based Attacks. Fig. 1 The Internet protocol stack that are inserted at different levels of the initial stack. However, most of the general public computers rarely use these more secured protocols and are therefore vulnerable to very simple header injection attacks (One should, however, notice that even recent security-oriented protocols may be prone to serious flaws [2]).

The case of TCP Reset attack is very representative of a header attack exploiting an important property of the connected transport layer TCP. TCP is used when a virtual connection is required. It is an efficient way of reliably exchanging big amounts of data between distant computers. A flag of the TCP header, the reset (RST) flag, is dedicated to the managing of a remote crash happening during the exchange. It abruptly terminates a connection, allowing a remote machine to flush its output buffers, and making it discarding further received packets. However necessary this mechanism may be, it can be easily perverted: a third-party computer may forge a false RST TCP packet and send it to the target, tricking it into closing the legitimate connection. The packet must of course contain a RST flag, but also other necessary values in order to appear genuine.

This is an example of a header attack exploiting a protocol vulnerability at the transport level. However, with the growing number of application-level protocols using the same header mechanisms, this kind of attack can also be implemented with application-level communication.

Header Attacks at the Application Layer Level

Applications relying on remote data exchange to work also rely on the separation between SDI and PCI. For all practical purposes, the application data as represented on Fig. 1 can itself be divided into "Application SDI" and "Application PCI." Thus, when no authentication mechanism is implemented, the application is vulnerable to header attacks targeting this application-level SDI. Among the many recent Web-oriented applications and application-level protocols, HTTP and SOAP (Simple Object Access Protocol) are two enlightening examples of easy-to-implement header attacks.

In the case of HTTP, HTTP header injections are a class of attacks relying on the dynamic generation of HTTP headers based on user input. When the application environment is unable to properly sanitize the strings that users are feeding to it, carefully chosen strings may lead it to turn data into valid commands or markup language excerpts. An attacker can then escape the data context and penetrate the surrounding framework. Once this is done, many doors are open that may lead to cross-site scripting [3] (allowing, for example, to bypass access control), cookie theft, Server Side Includes tags insertion, HTML insertion, etc.

SOAP is a protocol specification for exchanging generic structured information in the context of Web Services. It relies on other application-level protocols like RPC and HTTP to provide message transmission, and is based on XML (eXtensible Markup Language) to format its messages. SOAP messages carry information from one Web Service to another. In this context, driven by security requirements, a Web Service Security specification emerged and, in order to ensure authenticity as well as non-repudiation, a signature mechanism was provided based on the XML Digital Signature specification. The weakness resides in two points. First, the latter specifies that a message may contain one or more signed elements. Second, an XML Signature allows a non-contiguous object from an XML dataset to be signed separately. However, the signed object is referenced by an indirection that gives no information about the actual location of the signed element. This may lead to XML rewriting attacks [4] in which an object can be relocated and the signature still remains valid. These rewriting attacks may have many serious consequences: a SOAP message may be successfully replayed in different contexts, a transparent redirection toward a third party malicious Web Service may be made, timestamp elements may be circumvented, etc.

As Web Services – and other application-level infrastrucures – are probably destined to become major actors of future global information landscapes, header attacks must be taken into account very seriously. Fostered initially by technical choices made long ago, they continue to be on the agenda as more modern protocols are devised.

Applications

The possible consequences of this class of attacks are numerous. As seen with application-based header attacks, these offensives may lead to arbitrary redirections, data modification or access, and privilege escalation.

When put in place at low-level layers, they may help implement identity theft, bypass filtering tools by allowing hidden communication channels, denial of service [5], and even utterly sever the connection between remote computers.

Recommended Reading

1. Braden R (ed) (1989) Request for comments #1122: requirements for internet hosts – communication layers. Internet Engineering Task Force, Network Working Group
2. Marlinspike M (2009) More tricks for defeating SSL. In: Proceedings of the 2009 technical security conference, Las Vegas. http://www.blackhat.com/html/bh-usa-09/bh-usa-09-archives.html

3. Martin M, Lam MS (2008) Automatic generation of XSS and SQL injection attacks with goal-directed model checking. In: Seventeenth USENIX security symposium, San Jose
4. Rassadko N (2006) Policy classes and query rewriting algorithm for XML security views. In: Damiani E, Liu P (eds) Data and applications security. Lecture Notes in Computer Science, vol 4721. Springer, Heidelberg
5. Shields C (2002) What do we mean by network denial of service? In: Proceedings of the 2002 IEEE workshop on information assurance and security, West Point, 17–19 June 2002

HEC Acronym is Often Used for Hyper Elliptic Curves

►Hyperelliptic Curves

Heredity

►DNA

High Assurance Evaluation Methods

►Formal Methods in Certification and Evaluation

Higher Order Derivative Attack

►Cube Attack

Hippocratic Database

Tyrone Grandison[1], Kristen LeFevre[2]
[1]Intelligent Information Systems, IBM Services Research, Hawthorne, NY, USA
[2]Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA

Synonyms
Privacy-aware database; Privacy-enabled database

Related Concepts
►Data Retention

Definition
A database system that has privacy (enablement) as its fundamental goal.

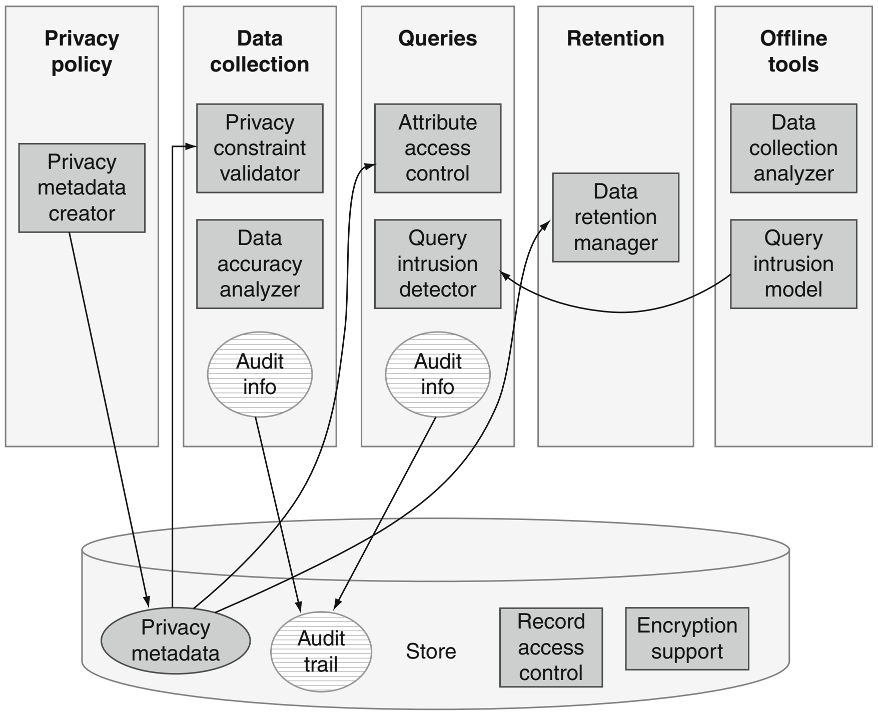
Background
The term "Hippocratic Database" (HDB for short) was coined in 2002 in a paper by Agrawal, Kiernan, Srikant, and Xu at the Very Large Databases (VLDB) conference [1]. They observed that technology trends, including the World Wide Web, were leading to a marked increase in electronic collection and storage of private and personal information. The HDB vision (Fig. 1) provided insight into a new class of data systems – one that required the redesign of current systems and that enabled the data system to automatically manage sensitive information without impeding information flow.

Theory
HDB is inspired by the privacy provision of the Hippocratic Oath, which states that:

► . . . about whatever I may see or hear in treatment, or even without treatment, in the life of human beings – things that should not ever be blurted outside – I will remain silent, holding such things to be unutterable [2].

HDB was not intended to be a fixed technology set. Instead, it expressed a vision for a class of database systems, tools, and applications that are designed with the primary goal of storing and managing personal and private data. In [1], they outlined ten key principles that should be incorporated into the design of future database systems. The following represents our interpretation of these ten principles, and examples of instances where the principles have been applied.

1. *Purpose Specification*: When personal information is collected, the purpose(s) for which the data is collected should be permanently associated with that in formation. For example, if an individual's address is collected for the purposes of shipping an order, this information should be maintained by default (e.g., as a form of meta data). This principle has been applied, for example, in the development of purpose based access control [3, 5].
2. *Consent*: Similarly, the user should provide consent for each purpose for which her data is collected. For example, if the user's address is collected for the purpose of shipping, then additional consent should be obtained before using this information for another



Hippocratic Database. Fig. 1 The vision of a Hippocratic Database (Diagram from [1])

purpose. Several techniques for enabling database consent management have been proposed [6, 7]. This issue is likely to persist as the debate over enabling the *secondary use of data* gains prominence, especially in sectors like health care.

3. *Limited Collection*: The personal data collected should be the minimum necessary for the required tasks [8].
4. *Limited Use*: Similar to consent, personal data should only be used for the purpose for which it was collected. Ideas on enforcing limited use are emerging in the field. For example, Angela C. Duta and Ken Barker [9] provide an example of how it could be done for XML data.
5. *Limited Disclosure*: Personal information should not be communicated outside the database for purposes other than those for which there is consent. From a technical perspective, this is one of the more heavily studied principles. Interestingly, the idea of limited disclosure can be interpreted in two distinct ways. On one hand, it can be interpreted literally (i.e., as a form of access control [4, 5]). Alternatively, the idea of limited disclosure can also be equated with statistical disclosure prevention [10–12], in which personal data can be used for computing aggregate statistics, as long as the underlying data cannot be inferred.
6. *Limited Retention*: Personal data should not be retained beyond the period necessary for fulfilling the purpose(s) for which it was collected. From a technical perspective, securely deleting data from a database can be a nontrivial task [13, 14]; it is important to guarantee that the data is securely removed from all parts of the system.
7. *Accuracy*: Personal data stored in the database should be accurate and up to date. Individuals should have the opportunity to correct inaccuracies.
8. *Safety*: This principle is most closely connected to conventional notions of security. Essentially, data stored in the database should be protected from theft and other security vulnerabilities.
9. *Openness*: Related to accuracy and compliance, an individual should be able to access his or her personal data stored in the database. The idea of openness can also be interpreted to include the idea that an individual should be able to find out how her data has been used (e.g., [15–17]).
10. *Compliance*: An individual who contributes personal data to a database should be able to verify all of the above principles. From a regulatory perspective, an authorized auditor should also be able to verify compliance with legal regulations pertaining to data use [18], e.g., HIPAA, Sarbanes-Oxley, etc.

Applications

HDB was initially applied to SQL (Structured Query Language) statements in the context of relational databases. Further work has implemented HDB for non-relational systems and to DML (Data Manipulation Language) statements. The technology has been prototyped in the Health Care, Finance, Government, and Scientific Research domains.

Open Problems

1. Improved Policy Specification – For HDB controls to be completely effective, policies must accurately capture the data usage practices of enterprises and the preference and choices of individuals concerning the use and disclosure of their personal information. The policy language must be fine grained enough to allow enterprises to collect, use, and disclose the minimum necessary information to accomplish their intended purposes. It must also be simple enough that technically unsophisticated individuals can understand the consequences of their decisions to provide personal information.
2. Enforcement After Extraction – Current implementations of HDB are adept at limiting disclosure of information contained within the database, but does not exert any control or safeguards over information that is legitimately extracted and transferred outside of the database.
3. Filter and Deny Semantics – HDB policy enforcement uses query predicates to filter results in compliance with the applicable policy rules. The system transforms the query so that the database only returns information that is compliant with the user's authorization, the enterprise's privacy policy, and any individual choices. Prohibited values that are sought by the query are returned as null values. However, in some circumstances, this type of filtering may not be desirable because it may mislead the user into thinking that the prohibited values do not actually exist.
4. Query Intrusion Detection – It is desirable to have the ability to detect illegitimate access by comparing a query access pattern to the usual and expected access pattern for that particular user and purpose. This capability advances the HDB safety principle by preventing inappropriate access through legitimate channels.
5. Data Integrity – An HDB system should provide guarantees on the soundness of the data that it contains. Individuals should have access to view and verify the accuracy of their information. Data cleansing can be used to identify and correct erroneous data. Maintaining the provenance of information can also provide an indication of the reliability of the information.

Recommended Reading

1. Agrawal R, Kiernan J, Srikant R, Xu Y (2002) Hippocratic databases. In: Proceedings of the international conference on very large data bases (VLDB), Hong Kong, 2002
2. Von Staden H (trans) (1966) In a pure and holy way: personal and professional conduct in the hippocratic oath. J His Med Appl Sci 51:406–408
3. Byun J-W, Bertino E, Li N (2005) Purpose based access control for complex data for privacy protection. In: Proceedings of the ACM symposium on access control models and technologies. ACM press, New York, pp 102–110
4. Ghazinour K, Majedi M, Barker K (2009) A lattice-based privacy aware access control model. In: Proceedings of the 2009 international conference on computational science and engineering. IEEE Computer Society, Washington, DC, pp 154–159
5. Al-Fedaghi S (2007) Beyond purpose-based privacy access control. In: Proceedings of the 18th Australasian database conference, Ballarat, Australia, January 29–February 2, 2007
6. LeFevre K, Agrawal R, Ercegovac V, Ramakrishnan R, Xu Y, DeWitt D (2004) Limiting disclosure in Hippocratic Database. In: Proceedings of the international conference on very large data bases (VLDB), Toronto, 2004
7. Agrawal R, Bird P, Grandison T, Kiernan J, Logan S, Rjaibi W (2005) Extending relational database systems to automatically enforce privacy policies. In: Proceedings of the 21st international conference on data engineering, Tokyo, 2005
8. Crépin L, Vercouter L, Jaquenet F, Demazeau Y, Boissier O (2008) Hippocratic multi-agent systems. In: Proceedings of the 10th international conference of enterprise information systems (ICEIS), Springer, Barcelona, pp 301–308
9. Duta AC, Barker K (2008) P4A: a new privacy model for XML. In: Proceedings of the 2008 data and applications security XXII. Springer, pp 65–80
10. Dwork C (2006) Differential privacy. In: Proceedings of 33rd international colloquium on automata, languages and programming, http://research.microsoft.com/en-us/projects/databaseprivacy/dwork.pdf, pp 1–12
11. Adam N, Wortmann J (1989) Security-control methods for statistical databases: a comparative study. ACM Comput Surv 21(4):515–556
12. Chen B-C, Kifer D, LeFevre K, Machanavajjhala A (2009) Privacy-preserving data publishing. Found Trends Databases 2(1–2):1–167
13. Stahlberg P, Miklau G, Levine B (2007) Threats to privacy in the forensic analysis for database systems. In: Proceedings of the ACM SIGMOD international conference on management of data, Beijing, 2007
14. Ananthanarayanan R, Gupta A, Mohania M (2008) Towards automated privacy compliance in the information Life cycle. In: Proceedings of the 2009 IFIP advances in web semantics, pp 247–259
15. Agrawal R, Bayardo R, Faloutsos C, Kiernan J, Rantzau R, Srikant R (2004) Auditing compliance in a Hippocratic Database. In: Proceedings of the international conference on very large data bases (VLDB), Toronto, 2004
16. Agrawal R, Evfimievski A, Kiernan J, Velu R (2007) Auditing disclosure by relevance ranking. In: Proceedings of the ACM

SIGMOD international conference on management of data, Beijing, 2007
17. Motwani R, Nabar S, Thomas D (2008) Auditing SQL queries. In: Proceedings of the international conference on data engineering (ICDE), Istanbul, 2008
18. Johnson CM, Grandison T (2007) Compliance with data protection laws using Hippocratic Database active enforcement and auditing. IBM Syst J 46(2):255–264

History-Based Separation of Duties

►Separation of Duties

HMAC

Bart Preneel
Department of Electrical Engineering-ESAT/COSIC, Katholieke Universiteit Leuven and IBBT, Leuven-Heverlee, Belgium

Synonyms

Hash-based message authentication code

Related Concepts

►Hash Functions; ►MAC Algorithms

Definition

HMAC is a ►MAC algorithm that is computed by two calls to a ►hash function; the calls have the secret key of the MAC algorithm as part of the data input.

Background

HMAC was designed by Bellare, Canetti, and Krawczyk [2] in 1996. The HMAC construction became popular because in the mid to late 1990s no secure and efficient custom designed MAC algorithms were available and hash functions (such as ►MD5) offered a much better software performance than block ciphers; as an example, HMAC based on ►MD5 is about ten times faster than ►CBC-MAC based on ►DES.

Theory

HMAC is constructed starting from an iterated hash function h that processes inputs in blocks of n bits:

$$\mathrm{HMAC}_K(x) = h((K \oplus opad) \| h((K \oplus ipad) \| x)) \,.$$

Here K is the key of the MAC algorithm (padded with zeroes to a block of n bits) and *opad* and *ipad* are constant n-bit strings (obtained by repeating the hexadecimal values "36_x" and "$5c_x$," respectively). The resulting MAC value can (optionally) be truncated. In practice, one can apply the compression function f of the hash function to the strings $K \oplus ipad$ and $K \oplus opad$, respectively, and store the resulting values as initial values for the inner and outer hashing operations. This will reduce the number of operations of the compression function by two. HMAC offers the advantage that it can be implemented without making any modification to the code of the hash function itself.

A variant of HMAC is NMAC: NMAC replaces the initial value (IV) of the hash function by a secret key. Denote by h_K a hash function with the IV replaced by the key K, then

$$\mathrm{NMAC}_{K_1\|K_2}(x) = h_{K_2}(h_{K_1}(x)) \,.$$

Bellare has improved the original security reduction of [2]; in [1] he showed that NMAC is a ►pseudo-random function and thus a secure MAC algorithm if the compression function is a ►pseudo-random function; for the security proof of HMAC with a single key as described above, pseudo-randomness under a related key attack is required. For NMAC the conditions can be further relaxed.

The best known generic attack on HMAC/NMAC is a forgery attack based on internal collisions [9]: it requires $2^{n/2}$ known text-MAC pairs (and a similar number of chosen texts if truncation is applied) and a single chosen text. A generic key recovery attack requires $2^{n/2}$ known text-MAC pairs and 2^{n+1} time, hence it is only meaningful for NMAC and for variants of HMAC that would use two different keys.

It turns out that the widely used hash functions ►MD4, ►MD5, and ►SHA-1 have weaknesses that result in more efficient attacks; the number of known or chosen text-MAC pairs is indicated as the data complexity. For HMAC-MD4 and NMAC-MD4, the following attacks are known: a forgery attack with data complexity 2^{58} [3] (compared to 2^{64} for a generic attack), and key recovery attacks with data complexity 2^{88} and time complexity 2^{95} [5] and data complexity 2^{72} and time complexity 2^{77} [11]. For NMAC-MD5, a forgery attack is known with data complexity 2^{47}; in the related key model, key recovery attacks exist with data complexity 2^{51} and time complexity 2^{100} [5, 10] and with data complexity 2^{75} and time complexity 2^{75} [11]; extending these attacks to HMAC is non-trivial. For HMAC-SHA-1 and NMAC-SHA-1 reduced to 34 out of 80 steps, a forgery with data complexity 2^{32} exists; a (partial) key recovery attack has been found for 53 out of 80 steps with a data complexity of $2^{97.5}$ [10]. Results are also known in a different attack model, in which one attempts to distinguish HMAC/NMAC based on a particular hash function from

HMAC/NMAC based on a hash function with an ideal compression function [7, 10].

Applications

HMAC has been included in ISO/IEC 9797-2 [6] and in FIPS 198 [4]. It is also widely used on the Internet after its standardization in RFC 2104 [8]: HMAC-SHA-1 with a 160-bit key and a truncation to 96 bits is the mandatory algorithm for providing message authentication at the network layer (►IPsec), and HMAC-MD5 is optional. ►TLS also uses both algorithms.

As ►AES implementations are faster than ►SHA-1 and ►SHA-2, HMAC no longer has a performance advantage; it is unclear whether this will change because of the ►AHS competition (SHA-3). A limitation of ►AES-based MAC algorithms is that forgery attacks exist with data complexity of 2^{64}, while for HMAC-SHA-1 the best known forgery attack has data complexity 2^{80}; for HMAC-SHA-2 the security level is at least 2^{128}.

Recommended Reading

1. Bellare M (2006) New proofs for new proofs for NMAC and HMAC: security without collision-resistance. In: Dwork C (ed) Advances in cryptology – CRYPTO '06: proceedings, Santa Barbara, 20–24 August 2006. Lecture notes in computer science, vol 4117. Springer, Berlin, pp 602–619. Full version http://eprint.iacr.org/2006/043
2. Bellare M, Canetti R, Krawczyk H (1996) Keying hash functions for message authentication. In: Koblitz N (ed) Advances in cryptology – CRYPTO '96: proceedings, Santa Barbara, 18–22 August 1996. Lecture notes in computer science, vol 1109. Springer, Berlin, pp 1–15. Full version http://www.cs.ucsd.edu/users/mihir/papers/hmac.html
3. Contini S, Yin YL (2006) Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions. In: Lai X, Chen K (eds) Advances in cryptology – ASIACRYPT '06: proceedings, Shanghai, 3–7 December 2006. Lecture notes in computer science, vol 4284. Springer, Berlin, pp 36–53
4. FIPS 198 (2002) The keyed-hash message authentication code (HMAC). NIST, US Department of Commerce, Washington DC, Gaithersburg
5. Fouque P-A, Leurent G, Nguyen PQ (2007) Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In: Menezes A (ed) Advances in cryptology – CRYPTO '07: proceedings, Santa Barbara, 19–23 August 2007. Lecture notes in computer science, vol 4622. Springer, Berlin, pp 13–30
6. ISO/IEC 9797 (2002) Information technology – Security techniques – Message Authentication Codes (MACs), Part 2: Mechanisms using a dedicated hash-function
7. Kim J, Biryukov A, Preneel B, Hong S (2006) On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1. In: De Prisco R, Yung M (eds) Security and cryptography for networks (SCN 2006), Maiori, 6–8 September 2006. Lecture notes in computer science, vol 4116. Springer, Berlin, pp 242–256
8. Krawczyk H, Bellare M, Canetti R (1997) HMAC: keyed-hashing for message authentication. RFC 2104, February 1997
9. Preneel B, van Oorschot PC (1995) MDx-MAC and building fast MACs from hash functions. In: Coppersmith D (ed) Advances in cryptology – CRYPTO '95: proceedings, Santa Barbara, 27–31 August 1995. Lecture notes in computer science, vol 963. Springer, Berlin, pp 1–14
10. Rechberger C, Rijmen V (2007) On authentication with HMAC and non-random properties. In: Dietrich S, Dhamija R (eds) Financial cryptography and data security. Lecture notes in computer science, vol 4886. Springer, Berlin, pp 119–133. Full version http://eprint.iacr.org/2006/290
11. Wang L, Ohta K, Kunihiro N (2008) New key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5. In: Smart NP (ed) Advances in cryptology – EUROCRYPT '08: proceedings, Istanbul, 13–17 April 2008. Lecture notes in computer science, vol 4965. Springer, Berlin, pp 237–253

Homomorphic Encryption

Berry Schoenmakers
Department of Mathematics and Computer Science, Technische Universiteit Eindhoven, Eindhoven, The Netherlands

Related Concepts

►Threshold Homomorphic Cryptosystems

Definition

An encryption mechanism E is called homomorphic basically if it preserves certain algebraic structure between the plaintext space and the ciphertext space, where the encryption key is fixed. For example, if the product of any two ciphertexts is equal to a ciphertext of the sum of the two corresponding plaintexts: $E(m_1) * E(m_2) = E(m_1 + m_2)$, where it is understood that all these encryptions use the same key.

Background

Homomorphic properties are natural for number-theoretic public key cryptosystems like ElGamal, Paillier, and also plain RSA. Homomorphic cryptosystems are routinely used in the design of privacy-protecting cryptographic schemes, such as secret ballot election schemes and sealed bid auction schemes. A typical use is to compute a random re-encryption $c' = E(m, r')$ of a given probabilistic encryption $c = E(m, r)$ such that ciphertexts c and c' contain the same plaintext m, but the randomness r' is statistically independent of r. By presenting two re-encrypted ciphertexts in random order one cannot tell which one is which even with access to the original ciphertexts.

Homomorphic cryptosystems are secure against passive attacks (chosen-plaintext attacks), hence satisfy computational indistinguishability. No security against active attacks (adaptive chosen-ciphertext attacks) is provided, as homomorphic encryptions are trivially malleable.

Theory

Notions of homomorphic encryption often correspond directly to mathematical notions of homomorphism, which are mappings that preserve certain algebraic structure. For instance, when encryption $E : M \rightarrow C$ (for a fixed key) is viewed as a function and both the plaintext space M and the ciphertext space C are endowed with a group structure, E is homomorphic if it is a group homomorphism from M to C. More generally, for probabilistic encryption $E : M \times R \rightarrow C$, the randomness space R should be endowed with a group structure as well, such that E is a group homomorphism from the direct product group $M \times R$ to C.

The well-known cryptosystems RSA, ElGamal, and Paillier are homomorphic in the following sense. Plain RSA encryption is multiplicatively homomorphic as for any $m_1, m_2 \in \mathbb{Z}_n^*$:

$$\left(m_1^e \bmod n\right) * \left(m_2^e \bmod n\right) = \left(m_1 m_2\right)^e \bmod n.$$

Similarly, ElGamal encryption is multiplicatively homomorphic as for any $m_1, m_2 \in \langle g \rangle$:

$$(g^{r_1}, y^{r_1} m_1) * (g^{r_2}, y^{r_2} m_2) = (g^{r_1+r_2}, y^{r_1+r_2} m_1 m_2),$$

where $r_1, r_2 \in_R \mathbb{Z}_q$ with $q = \operatorname{ord}(g)$. Finally, Paillier encryption is additively homomorphic as for any m_1, $m_2 \in \mathbb{Z}_n$:

$$\left(g^{m_1} r_1^n \bmod n^2\right) * \left(g^{m_2} r_2^n \bmod n^2\right) = \left(g^{m_1+m_2} (r_1 r_2)^n \bmod n^2\right),$$

where $r_1, r_2 \in_R \mathbb{Z}_n^*$.

Many more homomorphic cryptosystems have been introduced in the literature. Goldwasser–Micali encryption is based on quadratic residues and is XOR-homomor phic [10]; this was extended in [3] to higher-order residues. The additively homomorphic variant of ElGamal [5] is often used instead of multiplicatively homomorphic ElGamal. Generalizations of Paillier's cryptosystem are additively homomorphic over a plaintext space of flexible size [6]. Also, pairing-based homomorphic encryption schemes were introduced [2], and these schemes are not just additively homomorphic but at the same time multiplicatively homomorphic to a limited degree.

Fully homomorphic encryption is a strictly more powerful type of homomorphic encryption, which relates to the mathematical notion of a ring homomorphism. A finite ring, and in particular a finite field, involves two operations like addition and multiplication. Fully homomorphic encryption facilitates both addition and multiplication of encrypted values, which implies that any computable function can be evaluated on encrypted values solely with knowledge of the public key. So, given an n-ary function f and ciphertexts $E(x_1), \ldots, E(x_n)$, one is able to compute efficiently (i.e., in polynomial time in a security parameter) a ciphertext $E(f(x_1, \ldots, x_n))$. The existence of fully homomorphic encryption had been open for a long time until Gentry presented a lattice-based construction in 2009 [9], building on the progress in lattice-based cryptography since the Ajtai–Dwork public key cryptosystem appeared in 1997 [1]. A conceptually simple fully homomorphic cryptosystem, working over the integers, was introduced subsequently [7].

Applications

Homomorphic properties of public-key cryptosystems were recognized early on in the late 1970s, as well as the potential use for advanced privacy protection [11]. Homomorphic encryption can be used to construct fundamental primitives such as oblivious transfer, and is often used as a building block in secure multiparty computation, particularly in the two-party case. Homomorphic encryption is combined with threshold cryptography to yield threshold homomorphic cryptosystems, which are used in the construction of practical solutions for electronic elections, starting with [5].

More generally, secure multiparty computation for any (computable) function f can be based on threshold homomorphic cryptosystems [4, 8] by securely evaluating a circuit representing f. These general solutions require interaction between the parties for every secure multiplication gate – assuming an additively homomorphic cryptosystem is used. Using fully homomorphic encryption these interactions can be eliminated, providing a basic solution to the problem of noninteractive secure computation, once combined with threshold cryptography.

Recommended Reading

1. Ajtai M, Dwork C (1997) A public-key cryptosystem with worst-case/average-case equivalence. In: Proc. 29th symposium on theory of computing (STOC '97), ACM, New York, pp 284–293
2. Boneh D, Goh EJ, Nissim K (2005) Evaluating 2-DNF formulas on ciphertexts. In: Proc. 2nd theory of cryptography conference (TCC 2005), vol 3378. LNCS, Springer, Berlin, p 325
3. Cohen J, Fischer M (1985) A robust and verifiable cryptographically secure election scheme. In: Proc. 26th IEEE symposium on foundations of computer science (FOCS '85), IEEE Computer Society, pp 372–382
4. Cramer R, Damgård I, Nielsen JB (2001) Multiparty computation from threshold homomorphic encryption. In: Advances in cryptology – EUROCRYPT '01, vol 2045. LNCS, Springer, Berlin, pp 280–300. Full version eprint.iacr.org/2000/055, October 27, 2000

5. Cramer R, Gennaro R, Schoenmakers B (1997) A secure and optimally efficient multi-authority election scheme. In: Advances in cryptology – EUROCRYPT '97, vol 1233. LNCS, Springer, Berlin, pp 103–118
6. Damgård I, Jurik M (2001) A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In: Public key cryptography – PKC '01, vol 1992. LNCS, Springer, Berlin, pp 119–136
7. van Dijk M, Gentry C, Halevi S, Vaikuntanathan V (2010) Fully homomorphic encryption over the integers. In: Advances in cryptology – EUROCRYPT '10, vol 6110. LNCS, Springer, Berlin, p 24
8. Franklin M, Haber S (1996) Joint encryption and message-efficient secure computation. J Cryptol 9(4):217–232
9. Gentry C (2009) Fully homomorphic encryption using ideal lattices. In: Proc. 41st symposium on theory of computing (STOC '09), ACM, New York, p 169
10. Goldwasser S, Micali S (1984) Probabilistic encryption. J Com Syst Sci 28(2):270–299
11. Rivest R, Adleman L, Dertouzos M (1978) On data banks and privacy homomorphisms. In: Foundations of secure computation, Academic Press, pp 169–177

Homomorphism

Burt Kaliski
Office of the CTO, EMC Corporation, Hopkinton, MA, USA

Related Concepts

►Group; ►Homomorphic Encryption; ►Ring

Definition

A *homomorphism* is a mapping between two ►groups that respects the group structure.

Theory

Let $(S, \circ)$ and $(T, \bullet)$ be two groups, and let f be a mapping from S to T. The mapping f is a homomorphism if, for all $x, y \in S$,

$$f(x^{\circ}y) = f(x) \bullet f(y).$$

As an example, the mapping $f(x) = x^e \bmod n$ in ►RSA public-key encryption is a homomorphism since

$$f(xy) \equiv (xy)^e \equiv f(x)f(y) \pmod{n}.$$

In this case, the sets S and T and the operations $^{\circ}$ and $^{\bullet}$ are the same.

Another homomorphism is the mapping $f(x) = g^x \bmod p$ related to the ►discrete logarithm problem:

$$f(x+y) \equiv f(x)f(y) \pmod{p}.$$

Here, the sets and the operations are different between the two sides: S is the group of integers modulo $p-1$ under addition, and T is the subgroup of the multiplicative group modulo p generated by g.

►Ring *homomorphisms* that respect the structure of two operations may be defined similarly.

If the mapping f has an inverse g such that g is a homomorphism from T to S, then f is said to be an *isomorphism*.

Applications

Homomorphisms are important in cryptography since they preserve relationships between elements across a transformation such as encryption. Sometimes the structure enables additional security features (as in ►blind signatures), and other times it enables additional attacks (e.g., Bleichenbacher's attack on an unprotected form of RSA encryption [1]).

Recommended Reading

1. Bleichenbacher D (1998) Chosen ciphertext attacks against protocols based on RSA encryption standard PKCS #1". In: Krawczyk H (ed) Advances in cryptology – CRYPTO '98. Lecture notes in computer science, vol 1462. Springer, Berlin, pp 1–12

HRU

Vijay Alturi
Management Science and Information Systems Department, Center for Information Management, Integration and Connectivity, Rutgers University, Newark, NJ, USA

Related Concepts

►Access Control Policies, Models, and Mechanisms; ►Access Matrix

Background

The security model proposed by Harrison, Ruzzo, and Ullman (HRU) [1] is a discretionary access control model. In HRU, the current set of access rights at any given time can be represented by a matrix, with one row for each subject and one column for each subject and object (all subjects are also objects). Each cell in the table contains the list of access rights that the specific subject has for the particular object. In addition, HRU defines a primitive set of operations that can be performed.

Definition

The components of the HRU Model include:

- A set of subjects S
- A set of objects O
- A set of access rights R
- An access matrix $M = (M_{so})s \in S, o \in O$, the entry M_{so} is the subset of R specifying the rights subject s has on object o

The protection system is represented as a state machine where each state is defined as a triple triple (S, O, M). States are changed by altering the access matrix M. Change of the protection system is modeled by the following six primitive operations:

- Enter r into M_{so}
- Delete r from M_{so}
- Create subject s
- Create object o
- Destroy subject s
- Destroy object o

Commands in the HRU model have the format:

```
command $c(x_1, \ldots, x_k)$
  if $r_1$ in $M_{s_1,o_1}$ and
  if $r_2$ in $M_{s_2,o_2}$ and
  ...
  ...
  ...
  if $r_m$ in $M_{s_m,o_m}$
  then $op_1, \ldots, op_n$
end
```

In the above command, $s_1, \ldots, s_m$ and $o_1, \ldots, o_m$ are subjects and objects, respectively, that appear in the parameter list $c(x_1, \ldots, x_k)$. The condition part of the command checks whether particular access rights are present; the list of conditions can be empty. If all conditions hold, then the sequence of operations is executed. Each command contains at least one operation. Commands containing exactly one operation are said to be mono-operational commands, and those containing exactly one condition are said to be mono-conditional.

Theory

Safety analysis was first formalized in the HRU model. The safety question asks "Is there an algorithm for determining whether a protection system in some initial state is safe with respect to a right r?" This question can be formalized by the following two definitions of leakage of a right and unsafe state.

Definition 1 A state, that is, an access matrix M, is said to leak the right r if there exists a command c that adds the right r into an entry in the access matrix that previously did not contain r. More formally, there exist s and o such that $r \notin M_{so}$, and after the execution of c, $r \in M'_{so}$.

Definition 2 Given a protection system and a right r, the initial configuration Q_0 is said to be unsafe for r (or leaks r) if there is a configuration Q and a command α such that Q is reachable from Q_0 and α leaks r from Q. Q_0 is said to be safe for r if Q_0 is not unsafe for r.

The above definition states that a state of a protection system, that is, its matrix M, is said to be safe with respect to the right r if no sequence of commands can transform M into a state that leaks r.

Safety is undecidable in the general HRU model, which can be stated formally by the following theorem.

Theorem 1 Given an access matrix M and a right r, verifying the safety of M with respect to r is an undecidable problem.

The proof is based on mapping the safety problem to the halting problem of a Turing machine. Turing machines are general models of computing devices, expressed as a machine reading a tape that has a string of zeros and ones. The safety problem can be mapped to the problem of whether the Turing machine will ever halt when reading commands from the tape. Several decidable results about Turing machines are well known, including one that shows it is impossible to develop a general procedure to determine whether a given Turing machine will halt when performing a given computation. The proof of the above theorem follows by the demonstration that a decision procedure for protection systems would also solve the halting problem for Turing machines, which is known to be unsolvable.

However, it has been proven that some restrictive cases have decidable safety results. These include: (1) Safety for mono-operational systems is decidable but NP-Complete, and (2) Mono-conditional monotonic HRU is decidable.

Applications

This work has influenced the development of practical models and the analysis of several theoretical models developed subsequently.

Open Problems and Future Directions

The expressive power of a model is a measure of the flexibility to express policies for different requirements. Expressive power and manageable safety analysis are two conflicting properties of access control models. In general, the more expressive power a model has, the harder it is (if at all possible) to carry out safety analysis. To enhance the expressive power of access control models to suit to the different emerging environments, several extensions to the access control model have been proposed in recent years. Balancing the expressive power and safety in these extended access control models is still an open problem.

Recommended Reading

1. Harrison M, Ruzzo W, Ullman J (1976) Protection in operating systems. Commun ACM 19(8)461–471

HTTP Authentication

Jörg Schwenk
Horst Görtz Institute for IT Security, Ruhr University Bochum, Bochum, Germany

Synonyms

HTTP basic authentication; HTTP digest authentication

Related Concepts

►Challenge-and-Response Protocols; ►MD5; ►Password

Definition

Hypertext transfer protocol (*HTTP*) *authentication* [1] is an integration of password-based authentication, and of challenge-and-response-based authentication, into the HTTP [2]. Both basic concepts are modified to comply with the statelessness of the HTTP protocol.

Background

HTTP authentication was developed to protect access to static Web pages. It does not protect the confidentiality of the data during transport. Although it is still implemented in Web browsers and servers, alternative authentication mechanisms (mostly password-based) have gained in importance.

Applications

HTTP communication is based on a simple request–response scheme: The client (the Web browser) sends a request to the server, which answers with a three-digit status code and additional data. The client request specifies a HTTP method (mostly GET to retrieve data, and POST to send data), and the ressource on the server that should be accessed with this request. HTTP authentication is based on a shared secret between client and server (e.g., a password).

If the requested ressource has restricted access, the server can indicate that HTTP authentication is required by returning a special status code "401," which means "Authentication required." The server also indicates the type of authentication to be used, by sending the HTTP header line "WWW-Authenticate: Basic" to indicate password-based authentication, or "WWW-Authenticate: Digest" together with three additional parameters (as attribute values), to indicate challenge-and-response authentication. These three parameters are the domain of authentication (to help the client decide the shared secret to be used), the challenge value (from which the response will be computed), and a random nonce that will be returned unchanged, to protect against denial-of-service (DoS) attacks.

If "BASIC" authentication is requested, the client prompts the user to enter a password. This password is sent unencrypted (only Base64 coding is used) to the server. Thus "BASIC" authentication should only be used in conjunction with SSL/TLS to protoect the confidentiality of the password during transport.

To compute the response in "DIGEST" authentication, the client uses the hash function MD5 on at least the shared secret and the challenge, and on additional parameters (which depend on the different modes of operation), to compute a message authentication code (MAC). This MAC is sent back to the server, encoded as a string of hexadecimal numbers, as the value of the "response" attribute. The main extension to the standard challenge-and-response scheme results from the fact that the HTTP protocol is stateless: The server meanwhile has "forgotten" the challenge it had sent (to enhance HTTP perfomance), and thus the challenge has to be returned with the response, together with all other request parameters.

To complete authentication, the server in "BASIC" mode simply compares the password (or a hash of it) to a value stored in a database, or in a simple text file, and grants access if the two values match. In "DIGEST" mode, the server has to compute a second MAC based on the returned parameters, with the only difference that he uses his locally stored version of the shared secret. Again if the two MAC values match, access is granted.

Open Problems

The introduction of dynamic Web pages has triggered a change on the server architecture: The single Web server was replaced by multi-tier architectures consisting of a Web front end, an application logic, and a database server. Since the HTTP connection ends at the Web front end, HTML forms have replaced HTTP authentication for entering passwords: passwords can now be processed by the application logic, and used to get access to the database.

Authentication based on shared secrets does not scale well on the Internet: Shared secrets tend to have low entropy (allowing for dictionary attacks), and the same secrets are used on different links. Public key mechanisms for authentication therefore gain in importance. The SSL/TLS protocol has the ability to authenticate clients based on X.509 certificates; this feature was rarely used up to now, but is now adapted for modern browser-based computing paradigms.

Recommended Reading

1. Hallam-Baker P, Hostetler J, Lawrence S, Leach P, Luotonen A, Sink-E, Franks, J, Stewart L (1999) Http authentication: basic and digest access authentication. RFC 2617. http://www.ietf.org/rfc/rfc2617.txt, June 1999
2. Mogul J, Frystyk H, Masinter L, Leach P, Berners-Lee T, Fielding R, Gettys J (1999) Hypertext transfer protocol – http/1.1. RFC 2616. http://www.ietf.org/rfc/rfc2616.txt, June 1999

HTTP Basic Authentication

►HTTP Authentication

HTTP Cookie

►Cookie

HTTP Digest Access Authentication Scheme

►HTTP Digest Authentication

HTTP Digest Authentication

Italo Dacosta
Department of Computer Sciences, Georgia Institute of Technology, Paris, Atlanta, USA

Synonyms

Digest authentication; HTTP digest access authentication scheme

Related Concepts

►Authentication; ►Chosen Plaintext Attack; ►Hash Functions; ►HTTP; ►HTTP Authentication; ►HTTP Basic Authentication; ►MD5; ►Replay Attack

Definition

HTTP Digest Authentication is an application-layer, challenge-response authentication mechanism used by HTTP servers and proxies to verify the identity of users requesting access to protected resources.

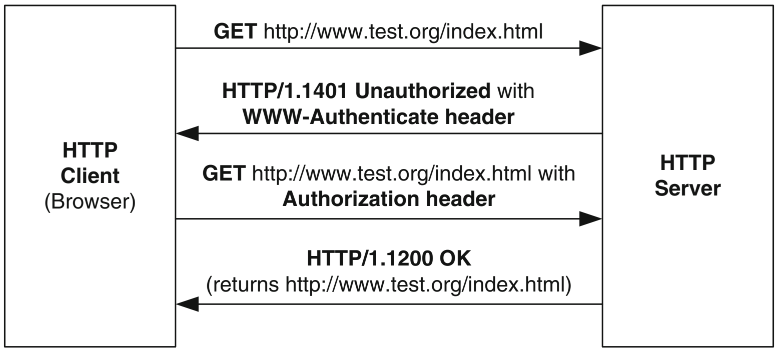
Background

The first authentication mechanism for HTTP was the Basic authentication scheme, defined in RFC 1945 (*Hypertext Transfer Protocol - HTTP/1.0*) in May 1996. Basic authentication is a simple mechanism but it has a significant security flaw: it sends users' passwords unprotected over the network. In response to this security weakness, the Digest authentication scheme was proposed in RFC 2069 (*An Extension to HTTP: Digest Access Authentication*) [1] in January 1997. RFC 2069 was later replaced by RFC 2617 (*HTTP Authentication: Basic and Digest Access Authentication*) [2] in June 1999. RFC 2617 solved some problems with the original Digest Authentication scheme and added new security enhancements to the scheme.

Applications

Digest and *Basic* authentication assume that each user shares a secret (password) with the server. The user needs to prove her knowledge of the password to the server in order to be able to access protected resources. For this purpose, *Basic authentication* requires the user to send her password unprotected (cleartext) over the network to the server. However, sending cleartext passwords over the network is a serious security flaw because an attacker can eavesdrop an authentication session and learn the user's password. *Digest authentication* avoids this problem by sending a *hash* of the password and other authentication parameters instead of the password itself. In this way, it is more difficult for an attacker to obtain the user's password by capturing her network traffic.

Figure 1 shows a standard Digest authentication exchange. First, a client (i.e., a web browser) sends a request to an HTTP server to access a protected web page. Then, the server determines that the client must be authenticated before granting access to the requested web page. Therefore, the server challenges the client by sending a *401 Unauthorized* status code message that includes the *WWW-Authenticate* header field (a HTTP proxy returns a *407 Proxy Authentication Required* status code message with a *Proxy-Authenticate* header field). After receiving the


GET http://www.test.org/index.html
HTTP/1.1401 Unauthorized with WWW-Authenticate header
HTTP Client (Browser)
GET http://www.test.org/index.html with Authorization header
HTTP Server
HTTP/1.1200 OK (returns http://www.test.org/index.html)

HTTP Digest Authentication. Fig. 1 HTTP Digest Authentication exchange

```
HTTP/1.1401 Unauthorized
WWW-Authenticate: Digest
        realm="testrealm@host.com",
        qop="auth,auth-int",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

HTTP Digest Authentication. Fig. 2 WWW-Authenticate header example (from RFC 2617)

```
Authorization: Digest
        username="Mufasa",
        realm="testrealm@host.com",
        nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
        uri="/dir/index.html",
        qop=auth,
        nc=00000001,
        cnonce="0a4f113b",
        response="6629fae49393a05397450978507c4ef1",
        opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

HTTP Digest Authentication. Fig. 3 Authorization header example (from RCF 2617)

challenge message, the client computes the response based on the information included in the challenge. Then, the client resends the request to the server but including an *Authorization* header field which contains the response and other required parameters. Finally, the server also computes the response and compares it with the one sent by the client. In addition, the server verifies other information such as the freshness of the client's response and that the request is for the same original resource. If the verification is successful, the server returns the requested web page with a *200 OK* status code. Otherwise, the server challenges the client again using the same or new challenge information.

An example of the *WWW-Authenticate* header is presented in Fig. 2. The first value corresponds to the mechanism used (Digest or Basic). The *realm* parameter defines the protection space and indicates to the user what username and password to use. The *qop* is an optional parameter indicating the *quality of protection* values supported by the server. Two values are supported: *auth* (authentication only) and *auth-int* (authentication with integrity protection). If *qop* is not present, the traditional Digest authentication mode defined in RFC 2069 is used. The *nonce* is a unique string generated by the server to protect against replay-attacks. Finally, the *opaque* is an optional string generated by the server that should be returned by the client unchanged in subsequent requests.

RFC 2617 also defines other optional parameters that can be included in the *WWW-Authenticate* header. The *domain* is a list of URIs also used to define the protection space. The *stale* is a flag indicating that a previous request from the client was rejected because the nonce was stale. The *algorithm* is a string indicating the algorithm used for hash operations (MD5 is the default algorithm). Finally, the *auth-param* directive is reserved for future extensions.

Figure 3 provides an example of the *Authorization* header created by the client to send the response to the server. The first value corresponds to the mechanism used (Digest or Basic). The *username* value represents the user's name in the specified protection domain (realm). The *uri* parameter indicates the URI used in the original request. The *qop* is an optional directive indicating the "quality of protection" that the client has applied to the message (must be one of the alternatives indicated by the server). The *nc* (nonce count) is an optional value that indicates how many requests the client has sent with the current client nonce value. The server can use this value to detect request replays (i.e., requests using the same *nc* value). The *cnonce* (client nonce) is an optional nonce value generated by the client for protection against chosen-plaintext attacks and to provide mutual authentication and limited message integrity protection. The *nc* and *cnonce* parameters are used only if the *qop* directive was specified by the server in the challenge message. The *response* value corresponds to the response computed by the client as described below. The parameters *realm*, *nonce*, and *opaque* are the same values sent in the challenge by the server (described in previous paragraphs). Finally, the *auth-param* is an optional directive for future extensions.

In addition, RFC 2617 defines a third header: the *Authentication-Info* header. This header is sent by the server to the client after a successful verification of the client's response. The main purpose of this header is to provide mutual authentication: the server has to prove its knowledge of the password. However, the practical use of this header is limited by the additional performance overhead it introduces (i.e., additional messages).

The *response* calculation varies slightly according to the value of the *qop* parameter. If *qop=auth*, the response is calculated as:

$$\boxed{\begin{array}{rl}\textbf{Response} & = H(H(\text{A1}) : \text{nonce} : \text{nc} : \text{cnonce} : \text{qop} : H(\text{A2}))\\ \textbf{A1} & = \text{username} : \text{realm} : \text{password}\\ \textbf{A2} & = \text{method} : \text{uri}\end{array}}$$

where *method* corresponds to the HTTP method used (i.e., GET), *password* is the user's password and *H()* is the cryptographic hash function to use. Digest authentication uses MD5 as the default hash algorithm. The values are concatenated using colons (:) as separators.

If *qop=auth-int*, then only *A2* changes:

$$\boxed{\textbf{A2} \quad = \text{method} : \text{uri} : H(\text{entity-body})}$$

where *H(entity-body)* is the hash of the HTTP entity-body (not the message body.) This value provides limited message integrity protection.

Finally, if *qop* is not present, then the response is calculated according to the traditional Digest authentication scheme (RFC 2069):

$$\boxed{\textbf{Response} \quad = H(H(\text{A1}) : \text{nonce} : H(\text{A2}))}$$

where *A1* and *A2* are similar to the case where *qop=auth*.

HTTP Digest authentication was designed as a more secure alternative to HTTP Basic authentication. Its main goal is to provide user authentication without sending users' passwords unprotected over the network. Besides user authentication and password confidentiality, it provides limited message integrity protection (only certain fields are protected). In general, Digest authentication is more efficient and easier to deploy than other security mechanisms (i.e., public key authentication, Kerberos, etc.).

However, Digest authentication is considered a weak security protocol when compared with other protocols such as SSL (Secure Socket Layer), TLS (Transport Layer Security), Kerberos, or IPsec (Internet Protocol security). Standard Digest authentication implementations do not provide message confidentiality, mutual authentication or integrity protection of the whole message. Therefore, they are vulnerable to eavesdropping, message spoofing, redirection, and Man-In-The-Middle (MITM) attacks. Digest authentication is also vulnerable to password brute force and dictionary attacks. Another possible area of concern is the use of MD5 as hash algorithm, given the multiple reported weaknesses of this algorithm. While the security of Digest authentication can be enhanced with the quality of protection (*qop*) options, these parameters are optional. Therefore, these options are not supported by all HTTP clients and servers.

Despite its weaknesses, Digest authentication is also used as the default authentication mechanism for the Session Initiation Protocol (SIP). Only minor modifications are required for the use of Digest authentication in SIP, as described in RFC 3261.

Finally, an alternative to HTTP Digest authentication is the use of Basic authentication in combination with other security protocols such as SSL, TLS or IPsec. Similarly, web forms can also be combined with these security protocols to provide HTTP user authentication.

Recommended Reading

1. Franks J, Hostetler J, Leach P, Sink E, Stewart L (1997) An Extension to HTTP : Digest Access Authentication, IETF Network Working Group RFC 2069 (Obsolete), http://tools.ietf.org/html/rfc2069
2. Franks J, Hallam-Baker P, Hostetler J, Lawrence S, Leach P, Luotonen A, Stewart L (1999) HTTP Authentication: Basic and Digest Access Authentication. IETF NetworkWorking Group RFC 2617, 1999, http://tools.ietf.org/html/rfc2617
3. David G, Brian T (2002) HTTP: the definitive guide, 1st edn. O'Reilly Media, Inc., Sebastopol, CA

HTTP Session Security

Andrew Clark
Information Security Institute, Queensland University of Technology, Brisbane, Queensland, Australia

Synonyms

Web session security

Related Concepts

►Cookie; ►Cross Site Scripting Attacks; ►HTTP Authentication and Authorization; ►HTTP Digest Authentication; ►HTTPS; ►Protocol Cookies; ►Session Hijacking Attacks; ►Transport Layer Security (TLS)

Definition

Hypertext transfer protocol *(HTTP) session security* refers to the security of a collection of related network transactions between an HTTP client (typically, but not always, a Web browser) and an HTTP server (often called a Web server). The compromise of an HTTP session may result in the loss of confidentiality, integrity, or availability of the data being transmitted between the client and the server, or data held by the client or the server.

Background

The HTTP [1] is a generic, stateless, application-layer communication protocol used to transmit data between a client and a server. It allows a client and a server to communicate using simple request–response interactions whereby an HTTP request message is initiated by the client and is followed by a corresponding HTTP response message from the server.

The first version of HTTP was developed in the early 1990s and at that time it was used mainly for distributing static Web content (such as simple HTML and image files). Such interactions do not require the server to maintain any state information from one request to the next, since each can be treated as an independent, unrelated message. While HTTP is still in wide use today, the Internet-based applications that rely upon it have become considerably more complex. In particular, most current HTTP application servers will maintain some state information about each client with which it is interacting. For example, an online shopping application server might store information about the items that a customer has added to a virtual shopping cart. Or, an online banking application server might store information about the identity of a customer it is interacting with (after the customer has completed some authentication process).

The state information maintained by an HTTP server will be associated with a corresponding *HTTP session* – the collection of (usually related) HTTP messages exchanged between a client and a server over a period of time. Typically, when the session is initiated (by the client), the server will generate a *session identifier* that is returned to the client in an HTTP response message. Each subsequent request transmitted by the client during that session will contain a copy of the *session identifier* so that the server can determine the state information to associate with the client. *HTTP Cookies* [2] are an example of a mechanism supported by most HTTP clients and servers for transmitting a session identifier in the *header* of HTTP messages.

The information accessible via many HTTP-based applications today can be considered very sensitive and therefore it is critical that the HTTP session be conducted securely. Weaknesses in the way HTTP clients and servers manage and conduct HTTP sessions have lead to a variety of different attacks.

Theory

In practice there are many challenges associated with securely managing and conducting an HTTP session (see [4] for a full treatment). Both the client and the server must be careful to ensure that sensitive session-related information is protected during active sessions and properly disposed of when sessions complete. Additionally, the communications between the client and the server must be conducted in a manner that prevents an intermediary from interfering with an active session.

Most sensitive applications will require that the HTTP communications be conducted using the ►transport layer security (TLS) protocol. TLS provides crytpographic authentication and message encryption mechanisms for protecting the communications between the client and the server. While TLS supports cryptographic authentication of the client (by the server), this feature is optional and in many instances TLS is only relied upon for authentication of the server (by the client) and encryption of all transmitted HTTP messages. HTTP itself provides only simple client authentication mechanisms (►HTTP Authentication and Authorization) and, in general, when client authentication is required (and TLS is not used for client authentication), then the server application will usually implement its own authentication mechanism.

Designers of server applications must be careful to ensure that the methods they employ for conducting an HTTP session and maintaining the associated state information are secure. For example, session identifiers should be *fresh* (they should not have been previously used) and *unpredictable*. Otherwise, the session identifier associated with an active session might be either obtained from a previous session or guessed, leading to ►session hijacking attacks or *session fixation attacks* (see [3] for a more detailed discussion). If the client is to be authenticated, then the server must carefully monitor the authentication process and ensure that access to sensitive information is granted only after the authentication process has successfully completed.

Designers of client applications must ensure that sensitive information associated with one session is not exposed while the user carries out another session with a (potentially malicious) Web site. ►Cross-site scripting attacks, are an example of a class of attack which can potentially reveal session identifiers (and other sensitive data) to malicious entities.

Open Problems

Designing, implementing, and deploying secure HTTP applications is particularly difficult due to the complex and distributed nature and modern information systems. Secure management of the underlying HTTP sessions is likewise difficult due the complicated interactions between the different protocol layers, in particular Transmission Control Protocol (TCP), TLS, HTTP and the application itself. Web services compound this problem by adding the Simple Object Access Protocol (SOAP) into the mix. Currently all of these layers are relied upon (by applications) to provide different security services. The interdependencies between these different protocol layers can lead to subtle, difficult to detect, vulnerabilities.

Recommended Reading

1. Fielding R, Gettys J, Mogul J, Frystyk H, Masinterand L, Leach P, Berners-Lee T (1999) Hypertext transfer protocol – HTTP/1.1. RFC 2616 (draft standard), June 1999. Updated by RFC 2817
2. Kristol D, Montulli L (2000) HTTP state management mechanism. RFC 2965 (proposed standard), October 2000
3. Murphey L (2005) Secure session management: preventing security voids in web applications. SANS Institute, Bethesda, Maryland January 2005. Revision 3
4. The Open Web Application Security Project (OWASP). A guide to building secure Web applications and Web services, 2.0 Black Hat edition, July 2005 wasp.org

HTTPS, HTTP over TLS

Torben Pedersen
Cryptomathic, Århus, Denmark

Related Concepts

►Secure Socket Layer (SSL); ►Transaction Layer Security (TLS)

Definition

HTTPS provides a mechanism for sending HTTP messages over a TLS secured connection rather than directly over TCP. A secure HTTP request is made using an URL of the type "https://. . ." instead of the "http://. . ." request used for ordinary HTTP. The default HTTPS port number is 443, as assigned by the Internet Assigned Numbers Authority.

Background

The usage of HTTP in various (shopping) applications over Internet as well as the need to transmit sensitive information (such as payment information) in many of these introduced the requirement to use HTTP securely. HTTP over SSL was created to allow sending HTTP messages over an SSL secured channel. Today, TLS, the successor of SSL, is normally used to secure the channel.

Theory

SSL and its successor TLS can be used to set up a secure channel between a client and a server. This channel offers confidentiality and server authentication and can optionally provide client authentication.

Thus HTTPS, see [1], is a two-step process in which security mechanisms and the necessary session keys in TLS are agreed initially. These session keys establish a secure tunnel during which the actual messages can be subsequently transmitted. The HTTPS server must have a certified public key ("►Certificate" and "►Public Key Cryptography"), which is used when exchanging the session keys (e.g., the client generates and encrypts a common secret under the public key of the server). Only a server having the private key corresponding to the public key in the certificate is able to recover the exchanged secret and obtain a common key with the requesting entity. By looking up the information in the server's certificate, the client can therefore be confident that it communicates with the right server and that only this server can see the contents of the messages transmitted over HTTPS.

In case of client authentication, the client must have a certified key pair as well. During initial key agreement in TLS, this key pair is used to authenticate the client by making a *digital signature*. Thus the server may base its decision to proceed on the identity of the client. In particular, this allows the server to control access to its services.

There are three important limitations of HTTPS. First of all HTTPS does not provide end-to-end security, but only secures the communication channel between the client and the server. In many web applications the messages are processed by a backend application. HTTPS does not provide any security between the web server receiving the request and the backend application. End-to-end security can be provided by adding security on the application level (e.g., as proposed in [2]), but this is outside the scope of HTTPS and must not be confused with HTTPS.

Secondly, the server is only authenticated to the extent that the client verifies the TLS certificate of the server.

Browsers can ensure that the certificate is valid (against a root of trust) and that it matches the address of the server. However, in case this verification fails browsers normally leave it to the user to decide what to do. This opens up the door for users to accept a certificate that they should not have accepted.

Thirdly, some attacks have used a server with a name (URL) very close to that of the attacked server. If the attacker gets a TLS certificate for this malicious server and lures the user into accessing it (e.g., using phishing), TLS will not help the user, who doesn't notice the small difference in the URL.

Recommended Reading

1. RfC2818: HTTP over TLS. http://www.rfc-editor.org/rfc.html
2. RfC2660: the secure hypertext transfer protocol. http://www.rfc-editor.org/rfc.html

Human Ear Biometrics

►Ear Shape for Biometric Identification

Human Ear Identification

►Ear Shape for Biometric Identification

Human Ear Recognition

►Ear Shape for Biometric Identification

Human Ear Verification

►Ear Shape for Biometric Identification

Hybrid Encryption

Kaoru Kurosawa
Department of Computer and Information Sciences, Ibaraki University, Hitachi-shi, Japan

Related Concepts

►Digital Signatures; ►Symmetric-Key Encryption Scheme

Definition

In a hybrid encryption scheme, a public-key encryption technique is used to encrypt a key K (KEM part) and a symmetric-key encryption technique is used to encrypt the actual plaintext m with the key K (DEM part).

Background

Shoup proved that a hybrid encryption scheme is secure against chosen ciphertext attack (CCA-secure) if KEM and DEM are both CCA-secure. Kurosawa and Desmedt showed a novel construction; their KEM is not CCA-secure, yet the whole scheme is. After that, other approaches have been proposed too.

Theory

A hybrid encryption scheme consists of two parts, a public-key encryption part called KEM (key encapsulation mechanism) and a symmetric-key encryption part called DEM (data encapsulation mechanism). A hybrid encryption scheme itself is a public-key encryption scheme whose public key and secret key are the same as in the KEM. Hence its security definitions are the same as those of public-key encryption schemes.

A public-key encryption scheme is said to be IND-CPA secure (secure in the sense of indistinguishability against chosen plaintext attack) if an adversary obtains (effectively) no information about m from the ciphertext C. It is said to be IND-CCA secure (secure in the sense of indistinguishability against chosen ciphertext attack) if an adversary obtains (effectively) no information about m from C even if she has access to the decryption oracle that returns a decryption for each query C'. Of course, she cannot query the target ciphertext C.

An advantage of hybrid encryption schemes is that a plaintext can be any bit string of any length. Another advantage is that it is sometimes easier to construct secure hybrid encryption schemes than the direct construction of public-key encryption schemes.

As the first example, consider the ElGamal encryption scheme. If the underlying group is defined over an elliptic curve, then a plaintext m must be a point of the elliptic curve in order that it is IND-CPA secure. Thus the plaintext space is restricted.

As the second example, consider RSA. Remember that a ciphertext is given by $C = m^e \bmod N$, where (N, e) is a public key. RSA is not IND-CPA secure because an adversary can see that $m = 0$ if $C = 0$, and $m = 1$ if $C = 1$. Yet it is easy to construct a provably secure hybrid encryption scheme under the RSA assumption as shown below. (The RSA assumption claims that it is hard to compute m from (N, e) and $C = m^e \bmod N$.)

Define RSA-KEM in such a way that the encryption algorithm chooses $r \in Z_N^*$ randomly, and computes $c_1 = r^e \bmod N$ and $K = H(r)$, where H is a hash function. It then outputs (c_1, K). Next let the DEM part compute $c_2 = m \oplus \mathsf{PRBG(K)}$, where $\oplus$ denotes bitwise XOR and PRBG denotes a pseudorandom bit generator. (This is essentially the one-time-pad.) Finally let the ciphertext be $C = (c_1, c_2)$. This hybrid encryption scheme is IND-CPA under the RSA assumption in the random oracle model. (H is modeled as a random oracle.)

Further modify the DEM part as follows. First let $K = K_e||K_m$, where $||$ denotes concatination. Next compute $c_2 = m \oplus \mathsf{PRBG}(K_e)$ and $c_3 = MAC_{K_m}(c_2)$, where MAC denotes a message authentication code (MAC). Finally let the ciphertext be $C = (c_1, c_2, c_3)$. This hybrid encryption scheme is IND-CCA secure under the RSA assumption in the random oracle model. Thus it is easy to construct a provably secure hybrid encryption scheme.

A practical PRBG can be obtained by using a counter mode, where the seed K is used as a key of the underlying block cipher, say AES. That is, $\mathsf{PRBG}(K) = AES_K((0)_2)||AES_K((1)_2)||AES_K((2)_2)||\cdots$, where $(i)_2$ denotes the binary representation of i. This construction is a pseudorandom bit generator if the underlying block cipher is a pseudorandom permutation.

Formally, KEM is a tuple of three algorithms $\mathsf{KEM} = (K_{\text{kem}}, E_{\text{kem}}, D_{\text{kem}})$. The key generation algorithm K_{kem} generates a pair $(pk, sk) \overset{R}{\leftarrow} K_{kem}(1^n)$, where pk is a public key, sk is a secret key, and n is a security parameter. The encryption algorithm E_{kem} takes a public key pk, and returns $(c_1, K) \overset{R}{\leftarrow} E_{\text{kem}}(pk)$, where c_1 is a ciphertext and K is a DEM key. The decryption algorithm D_{kem} takes a secret key sk and a ciphertext c_1, and returns $D_{\text{kem}}(sk, c_1)$, which is either a DEM key K or *reject*.

A KEM is said to be CPA-secure if (pk, c_1, K) is indistinguishable from (pk, c_1, K'), where K' is a random string. It is said to be CCA-secure if (pk, c_1, K) is indistinguishable from (pk, c_1, K') even if an adversary has access to the decryption oracle.

It is easy to prove that a hybrid encryption scheme is IND-CPA secure if the KEM is CPA-secure and the DEM is the one-time-pad. (That is, a ciphertext is $C = (c_1, c_2 = m \oplus \mathsf{PRBG}(K))$.) Indeed, since K is indistinguishable from a random string, c_2 works as the one-time-pad. Shoup [20] proved that a hybrid encryption scheme is IND-CCA secure if the KEM is CCA-secure and the DEM is the one-time-pad plus MAC. (More precisely, it is enough that DEM is CCA-secure [11].)

For example, RSA-KEM is CCA-secure under the RSA assumption in the random oracle model. Another CCA-secure KEM is obtained as follows. Let G be a cyclic group of a prime order p, and let g be a generator. That is, $G = \{1, g, g^2, \cdots, g^{p-1}\}$. The key generation algorithm chooses $x \overset{R}{\leftarrow} Z_p$, and computes $y = g^x$, where $Z_p = \{0, 1, \cdots, p-1\}$. It outputs y as a public key, and x as a secret key. The encryption algorithm chooses $r \overset{R}{\leftarrow} Z_p$, and computes $c_1 = g^r$ and $K = H(g^r, y^r)$. This KEM is CCA-secure under the gap Diffie–Hellman (the gap DH) assumption in the random oracle model. The gap DH assumption claims that it is hard to compute g^{ab} from (g, g^a, g^b) even if it is allowed to have access to the decisional Diffie–Hellman (DDH) oracle. For a query $(g^\alpha, g^\beta, g^\gamma)$, the DDH oracle tells whether $\gamma = \alpha \cdot \beta \bmod p$ or not.

On the other hand, it is much harder to construct IND-CCA secure public-key encryption schemes without random oracles. The first schemes were presented by Naor and Yung [18], Rackoff and Simon [19], and Dolev, Dwork, and Noar [12]. However, they were quite impractical. The first truly practical IND-CCA secure public-key encryption scheme was shown by Cramer and Shoup [9]. The security of this scheme is proved under the DDH assumption, where the DDH assumption claims that it is hard to distinguish (g, g^a, g^b, g^{ab}) from (g, g^a, g^b, g^c), where $a, b, c \overset{R}{\leftarrow} Z_p$. In [20], Shoup showed a CCA-secure KEM as a variant of the Cramer–Shoup public-key encryption scheme under the DDH assumption.

Kurosawa and Desmedt [17] showed a new KEM by modifying the KEM of Shoup [20]. Its major practical advantage is that it saves the computation of one exponentiation and produces shorter ciphertexts. This efficiency improvement is the result of a novel observation: previous hybrid encryption schemes were proven secure by proving that the KEM was CCA-secure. On the other hand, their KEM is not CCA-secure, yet the whole scheme is, assuming the DDH assumption.

Their security proof relies on the use of information theoretically secure components in the DEM construction (specifically the key derivation function (KDF) and the MAC). Gennaro and Shoup [13] showed an alternative proof that removes the need for information theoretically secure KDF and MAC, thereby effectively improving the efficiency and applicability of Kurosawa–Desmedt scheme.

In [13, 17], the authors were not able to prove or disprove the CCA-security of the Kurosawa–Desmedt KEM. Choi et al. [8] proved that the Kurosawa–Desmedt KEM is actually *not* CCA-secure.

An interesting question is then, what property does the Kurosawa–Desmedt KEM satisfy so that the whole hybrid encryption scheme is IND-CCA secure? Abe, Gennaro, and Kurosawa [1] gave a possible answer to this question. They formalized the notion of *Tag-KEM* in which the encryption algorithm takes an external input called a tag. (The encryption algorithm of KEM takes no input.) They defined the notion of CCA-security for Tag-KEMs and showed that the Kurosawa–Desmedt scheme can be "explained" as a CCA-secure Tag-KEM. The resulting hybrid encryption scheme is CCA-secure if Tag-KEM is CCA-secure and DEM is just one-time-pad. Abe et al. also showed several generic construction methods of CCA-secure Tag-KEM.

A different approach was taken by Hofheinz and Kiltz in [16]. They defined the notion of *constrained* CCA-security (CCCA-security) for a KEM, and showed that this yields a CCA-secure hybrid encryption scheme along with an authenticated encryption scheme, which includes the Kurosawa–Desmedt scheme as an example.

Cramer and Shoup showed that their original scheme in [9] was a special instance of a generic paradigm based on *hash proof systems* [10]. Due to this abstraction, new CCA-secure schemes can be built based on different computational assumptions, such as quadratic residuosity and N-residuosity mod N^2. Kurosawa–Desmedt KEM can be generalized to *hash proof systems* also.

Other Directions Without Random Oracle

Canetti, Halevi, and Katz [7] showed a general method for constructing an IND-CCA secure public-key encryption scheme from an identity-based encryption scheme (IBE) and a one-time signature scheme, where the IBE must be selective-ID IND-CPA secure. Boneh and Katz [4] showed a more efficient method by replacing the one-time signature scheme with a MAC along with an appropriate KEM. Based on this approach, Boyen, Mei, and Waters [5] showed a further better KEM by taking advantage of specific properties of Boneh–Boyen IBE [3]. The KEM is CCA-secure under the decisional bilinear Diffie–Hellman (BDH) assumption.

Hanaoka and Kurosawa [14] showed another generic method for constructing a CCA-secure KEM from a broadcast encryption scheme that satisfies "verifiability." Based on this approach, they showed a CCA-secure KEM under the computational Diffie–Hellman (CDH) assumption such that the size of ciphertexts is the same as that of Cramer–Shoup KEM while the size of public key is larger. The CDH assumption, which is weaker than the DDH assumption, claims that it is hard to compute g^{ab} from (g, g^a, g^b). They also explicitly showed a CCCA-secure KEM under the hashed Diffie–Hellman (HDH) assumption, which is as efficient as the Kurosawa–Desmedt scheme. The HDH assumption, which is weaker than the DDH assumption, claims that it is hard to distinguish $(g, g^a, g^b, H(g^{ab}))$ from (g, g^a, g^b, r), where H is a hash function and r is a random string.

Recommended Reading

1. Abe M, Gennaro R, Kurosawa K (2008) Tag-KEM/DEM: a new framework for hybrid encryption. J Cryptol 21(1):97–130
2. Bellare M, Rogaway P (1993) Random oracles are practical: a paradigm for designing efficient protocols. In: ACM conference on computer and communications security. ACM Press, New York, pp 62–73
3. Boneh D, Boyen X (2004) Efficient selective-ID secure identity-based encryption without random oracles. EUROCRYPT. Springer, Berlin, pp 223–238
4. Boneh D, Katz J (2005) Improved efficiency for CCA-secure cryptosystems built using identity-based encryption. CT-RSA. Springer, Berlin, pp 87–103
5. Boyen X, Mei Q, Waters B (2005) Direct chosen ciphertext security from identity-based techniques. In: ACM conference on computer and communications security. ACM Press, New York, pp 320–329
6. Canetti R, Goldreich O, Halevi S (2004) The random oracle methodology, revisited. J ACM 51(4):557–594
7. Canetti R, Halevi S, Katz J (2004) Chosen-ciphertext security from identity-based encryption. EUROCRYPT. Springer, Berlin, pp 207–222
8. Choi SG, Herranz J, Hofheinz D, Hwang JY, Kiltz E, Lee DH, Yung M (2009) The Kurosawa-Desmedt key encapsulation is not chosen-ciphertext secure. Inform Process Lett 109(16):897–901
9. Cramer R, Shoup V (1998) A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. CRYPTO'98, LNCS vol 1462. Springer, Berlin, pp 13–25
10. Cramer R, Shoup V (2002) Universal hash proofs and a paradigm for chosen ciphertext secure public key encryption. Euro-Crypt'02, LNCS vol 2332. Springer, Berlin, pp 45–64
11. Cramer R, Shoup V (2003) Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. SIAM J Comput 33:167–226
12. Dolev D, Dwork C, Naor M (1991) Non-malleable cryptography. STOC'91. ACM Press, New York, pp 542–552
13. Gennaro R, Shoup V A note on an encryption scheme of Kurosawa and Desmedt. IACR eprint archive http://eprint.iacr.org/2004/194
14. Hanaoka G, Kurosawa K (2008) Efficient chosen ciphertext secure public key encryption under the computational Diffie-Hellman assumption. ASIACRYPT. Springer, Berlin, pp 308–325
15. Herranz J, Hofheinz D, Kiltz E The Kurosawa-Desmedt key encapsulation is not chosen-ciphertext secure. IACR eprint archive http://eprint.iacr.org/2006/207
16. Hofheinz D, Kiltz E (2007) Secure hybrid encryption from weakened key encapsulation. CRYPTO, LNCS vol 4622. Springer, pp 553–571
17. Kurosawa K, Desmedt Y (2004) A new paradigm of hybrid encryption scheme. CRYPTO'04, LNCS vol 3152. Springer, Berlin, pp 426–442
18. Naor M, Yung M (1990) Public-key cryptosystems provably secure against chosen ciphertext attacks. In: Proceedings of the twenty second annual ACM symposium on theory of computing, in STOC'90. ACM Press, New York, pp 427–437
19. Rackoff C, Simon D (1991) Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack. CRYPTO'91, LNCS, vol 576. Springer, Berlin, pp 433–444
20. Shoup V (2000) Using hash functions as a hedge against chosen ciphertext attack. EuroCrypt'00, LNCS vol 1807. Springer, Berlin, pp 275–288

Hyperelliptic Curve Discrete Logarithm Problem (HECDLP)

►Hyperelliptic Curve Security

Hyperelliptic Curve Security

Roberto Avanzi[1], Nicolas Thériault[2]
[1]Ruhr-Universität Bochum, Bochum, Germany
[2]Departamento de Matemática, Universidad del Bío-Bío, Talca, Chile

Synonyms

Hyperelliptic curve discrete logarithm problem (HECDLP)

Related Concepts

►Discrete Logarithm Problem; ►Elliptic Curve Discrete Logarithm Problem

Definition

The hyperelliptic curve discrete logarithm problem is a special case of the ►discrete logarithm problem.

Background

Let H be a ►hyperelliptic curve defined over a ►finite field $\mathbb{F}_q$, and let $D_a \in \mathrm{Pic}^0_H(\mathbb{F})$ be a divisor of order n. Given $D_b \in \langle D_a \rangle$, the *hyperelliptic discrete logarithm problem* consists in finding an integer λ, $0 \le \lambda \le n-1$ such that $D_b = \lambda D_a$. The *hyperelliptic discrete logarithm problem* is a special case of the discrete logarithm problem in which the cyclic group G is represented by the group of divisors in the divisor class group of a hyperelliptic curve. The intractability of the hyperelliptic curve discrete logarithm problem for curves of small genus makes these of cryptographic interest, thus motivating ►hyperelliptic curve cryptography.

Theory

Attacks on the hyperelliptic curve discrete logarithm problem can be divided in three main categories:

- Generic methods, which apply to all groups (regardless of how the groups are obtained).
- Index calculus methods, which take advantage of the general structure of divisor class group. These attacks depend on the genus and the size of the field over which the curve is defined, but are mostly independent of the specific curve.
- Special-purpose algorithms, which work either for specific curves or for curves defined over specific types of fields. These include Weil and Tate pairing attacks, Weil descent, and other mappings between curves of different types.

Generic Methods

All the methods that can be used to solve the discrete logarithm problem for generic groups can also be used to solve the hyperelliptic curve discrete logarithm problem. As a result, the hyperelliptic curves that are considered for cryptography should have a group order that is either prime or has a very large prime factor, in order to avoid the subgroup attack of Pohlig and Hellman [15].

Pollard's ρ method [16] remains the fastest algorithm to solve the discrete logarithm problem in the Jacobian of most hyperelliptic curves of genus 2, but unlike the ►elliptic curve discrete logarithm problem, other attacks often have to be taken into account when determining the security requirement for other hyperelliptic curves.

Index Calculus Methods

Index calculus methods did not prove very successful to solve the elliptic curve discrete logarithm problem in general, except for elliptic curves defined over field extensions of small degrees. The situation is quite different for hyperelliptic curve, where it becomes one of the main concerns when establishing security requirements.

Curves of Large Genus

The first successful adaptation of ►index calculus algorithm to hyperelliptic curve Jacobians is due to Adleman, DeMarrais, and Huang [1]. They used irreducible divisors (sums of a point and all its images under the Frobenius endomorphism) of a certain degree to define their factor base, and principal divisors to search for smooth relations. The algorithm of Adleman, DeMarrais, and Huang was later refined by Enge and Gaudry [4, 5] to obtain a running time of

$$L_{q^g}\left(\frac{1}{2}, \frac{\left(\sqrt{2t+1}+1\right)}{\sqrt{t}}\right)$$
$$= \exp\left(\frac{\left(\sqrt{2t+1}+1\right)}{\sqrt{t}} \sqrt{\ln q^g}\sqrt{\ln\ln q^g}\right)$$

where t is a constant smaller than $g/\log_g q$, and $q \le L_{q^g}(1/2, 1/\sqrt{t})$.

More recently, Enge, Gaudry, and Thomé [6] obtained a significant improvement, going from an $L(1/2)$ to an

$L(1/3)$ algorithm. Given a curve of genus g with a defining equation of degree a in x and b in y and $g > (\ln q)^2$, the running time becomes

$$L_{q^g}\left(\frac{1}{3}, \sqrt[3]{\frac{64\kappa}{9}}\right) = \exp\left(\sqrt[3]{\frac{64\kappa}{9}\ln q^g(\ln\ln q^g)^2}\right)$$

where κ is a constant no smaller than ab/g. If every monomial $x^i y^j$ in the equation defining the curve satisfies $bi + aj \le ab$ (e.g., for $C_{a,b}$ curves), the κ term can be removed from the running time.

Curves of Small Genus

The first presentation of an index calculus algorithm capable of efficiently solving the discrete logarithm problem in the Jacobian of hyperelliptic curves is due to Gaudry [9]. By using the points of the curve that are defined over $\mathbb{F}_q$ as a factor base, and decomposing reduced divisors according to their finite support, it becomes possible to solve the discrete logarithm problem in time $\widetilde{O}(q^2)$. To find the smooth relations used in the index calculus algorithm, Gaudry uses a random walk, similar to that used in Pollard's ρ method. Gaudry's algorithm effectively improves on the running time of Pollard's ρ method for curves of genus 5 and higher. The direct result of this algorithm was to limit the genus of hyperelliptic curve considered for cryptographic applications to at most 4.

Since the all points over $\mathbb{F}_q$ are statistically equivalent in terms of their impact on the index calculus, it is possible to use only a subset of the points of the curve defined over $\mathbb{F}_q$ as the factor base. By introducing "single large prime" [19] and "double large primes" [11, 14] variations, it has been possible to reduce the running time of the index calculus algorithm. The fastest known index calculus variation known at present takes time

$$\widetilde{O}\left(q^{2-\frac{2}{g}}\right)$$

to solve the discrete logarithm problem in the Jacobian of an hyperelliptic curve of genus g.

For curves of genus 2, the (best possible) complexity of index calculus techniques reduces to that Pollard's rho method, but with a constant factor larger than the $\sqrt{\pi}/2$ of Pollard ρ. As a consequence, genus 2 curves defined over prime fields and fields of (large) prime extension degree are generally considered secure against index calculus attacks (for fields of small extension degree, see the next subsection).

Given the impact of these attacks on the security of the hyperelliptic curves, more attention is given to curves of genus 2 than those of genus 3 and 4. Recent implementation results show that genus 3 and 4 curves can still be of cryptographic interest in some cases (►Hyperelliptic Curves Performance), but special care must be taken when choosing the size of the field over which the curve is defined. To compare with curves that rely on the running time of Pollard's ρ method, better knowledge about the constant factors is required. Some progress has been made in this direction, showing that variations that use up to one large prime have constant factors relatively similar to that of Pollard's ρ [21]. However, it remains unclear how to exactly balance the comparison for the double large prime variation: at best it is known to be effective in small examples. Nevertheless, many authors prefer to assume that index calculus algorithms have the same constant factor as Pollard's ρ metod, at the risk of disadvantaging curves of genus 3 and 4 in a fair comparison (but making it possible to compare the efficiency of those cryptosystems).

A surprising development in the discrete logarithm problem for algebraic curves came with the presentation of an index calculus algorithm geared toward non-hyperelliptic curves of low genus by Diem [2, 3]. Although it had been assumed that the discrete logarithm problem would be at least as difficult for non-hyperelliptic curves as it is for hyperelliptic ones (since hyperelliptic curves are a special type of algebraic curves), Diem showed that it is in fact easier. By reintroducing the use of principal divisors as the source of smooth relations (instead of reduced divisors), as it is done for curves of high genus, Diem was able to obtain a running time of

$$\widetilde{O}\left(q^{2-\frac{2}{d-2}}\right)$$

where d is the degree of the equation defining the curve. In general, this can be upper-bounded by $\widetilde{O}(q^{2-2/(g-1)})$ for non-hyperelliptic curves of genus g. This had a significant impact on the security of non-hyperelliptic curves of genus 3, since their discrete logarithm problem could now be solved in time $\widetilde{O}(q)$ (these curves have equations of degree 4). Since this is essentially the same running time required by curves of genus 2, but with a significantly more costly group arithmetic, non-hyperelliptic curves of genus 3 are no longer considered for cryptographic applications.

Curves over Field Extensions

Gaudry's technique[10] to perform an index calculus algorithm on curves defined over field extensions $\mathbb{F}_{q^m}$ although mainly intended for the elliptic curve discrete logarithm problem, also applies naturally to curves of higher genus. By constructing the factor base using points over $\mathbb{F}_q$ (rather than $\mathbb{F}_{q^m}$), it becomes possible to treat the genus g curve over $\mathbb{F}_{q^m}$ as if it were a genus mg hyperelliptic curve defined over the field $\mathbb{F}_q$. By using a two large primes variation of index calculus to these factor bases, one obtains a running time of $\widetilde{O}(q^{2-2/mg})$.

This approach is particularly effective for curves of genus 2 over quadratic field extensions. In that case the complexity of the discrete logarithm decreases to $\widetilde{O}(q^{3/2})$, down from the $O(q^2)$ complexity of Pollard's ρ method.

For other combinations of genus and degree of the field extension, the practicality of this attack is still unclear (due to large constant factors), but it remains a concern. As a consequence, it is generally preferred to avoid small extension degrees when choosing curves that will be used for cryptographic applications.

Special-Purpose Algorithms

Some families of curves have been found to be susceptible to more specialized attacks than Pollard's ρ method and the various index calculus algorithms. These families of curves should generally be either avoided for cryptographic applications, or their specific security requirements should be taken into account – for example in the case of pairing-based cryptosystems.

Weil and Tate Pairing Attacks

The pairing-based attacks by Frey and Rück [7] and Menezes, Okamoto, and Vanstone [13] can also be applied to supersingular hyperelliptic curves, reducing the HECDLP to a discrete logarithm problem over the field $\mathbb{F}_{q^k}$ with k not too large. As in the case of elliptic curves, it is always possible to define such a pairing, but for most curves the value of k is too large for the problem to be tractable in the finite field (not to mention the cost of computing the pairing itself).

At the same time, the ability to compute efficient Weil and Tate pairings for some hyperelliptic curves [12] opens the possibility of using pairing-based cryptosystems built upon those groups.

Weil Descent

The Weil descent attack is another approach that can be carried over from the elliptic curve discrete logarithm problem to the hyperelliptic curve discrete logarithm problem. Only a few papers deal with Weil descent attacks on hyperelliptic curves [8, 18, 20]. For genera higher than 1, it would seem to be more difficult to perform a Weil descent attack, in the sense that the proportion of isomorphism classes of curves that are at risk is lower than in the elliptic curve case. However, the possibility of such an attack remains present, and it is therefore natural to tread carefully before using extension degrees where a Weil descent attack would be possible in the context of elliptic curves.

Curve Mappings

Following the realization that the discrete logarithm problem on non-hyperelliptic curves may be weaker than on hyperelliptic curves of the same genus, the question arose as to which hyperelliptic Jacobians could be mapped to non-hyperelliptic ones, and how this could be done.

The first (and so far only) work to present an answer is due to Smith [17]. It consists in a technique to map Jacobians of hyperelliptic curves of genus 3 in the real model to Jacobians of non-hyperelliptic ones (again of genus 3). The probability of success of this attack depends heavily on the factorization type of the equation defining the hyperelliptic curve (over the field $\mathbb{F}_q$).

The attack always produces a non-hyperelliptic curve of genus 3, but is successful only when the field over which the curve is defined remains the same. In that case, the discrete logarithm problem can be solved in time $\widetilde{O}(q)$ using Diem's attack. If the field over which the non-hyperelliptic curve is defined changes, the new field is an extension of $\mathbb{F}_q$, and the index calculus attack becomes slower than it was on the hyperelliptic curve (hence the attack fail).

The probability of success for a randomly chosen hyperelliptic curves defined over a field $\mathbb{F}_q$ is on average close to 20% (in fact $\approx$ 18.57%), but this varies wildly depending on the factorization type of the defining equation. If the polynomial $F(x)$ defining the curve (as $y^2 = F(x)$) splits over $\mathbb{F}_q$, success is almost certain (the probability of success is $1 - (3/4)^{105}$). On the other hand, if $F(x)$ contains an irreducible factor of degree 5 or 7, or has exactly one irreducible factor of degree 3, then the attack always fails. The remaining types of factorization have intermediate probabilities of success.

As a result of this attack, it is recommended not to choose the equation of a hyperelliptic curve of genus 3 completely at random, but to ensure that it factors in certain ways.

So far it is not known if the attack of Smith can be broadened to include a larger proportion of the genus 3 curves, or if it can be applied in even characteristic. It is not yet known how to do similar constructions for curves of higher genus.

Recommended Reading

1. Adleman LM, DeMarrais J, Huang MD (1999) A subexpotential algorithm for discrete logarithms over hyperelliptic curves of large genus over GF(q). Theor Comput Sci 226:7–18
2. Diem C (2006) Index calculus in class groups of plane curves of small degree. Algorithmic Number Theory Symposium – ANTS VII, Lecture Notes in Computer Science 4076:543–557. Springer, Berlin
3. Diem C, Thomé E (2008) Index calculus in class groups of non-hyperelliptic curves of genus three. J Cryptol 21:593–611
4. Enge A (2002) Computing discrete logarithms in high-genus hyperelliptic Jacobians in provably subexponential time. Math Comput 71(238):729–742
5. Enge A, Gaudry P (2002) A general framework for subexponential discrete logarithm algorithms. Acta Arithmet 102:83–103

6. Enge A, Gaudry P, Thomé E (2009) An L (1/3) discrete logarithm algorithm for low degree curves. preprint
7. Frey G, Rück H (1994) A remark concerning m-divisibility and the discrete logarithm problem in the divisor class group of curves. Math Comput 62(206):865–874
8. Galbraith SD (2003) Weil descent of Jacobians. Discrete Appl Math 128(1):165–180
9. Gaudry P (2000) An algorithm for solving the discrete logarithm problem on hyperelliptic curves. Advances in cryptology – EUROCRYPT 2000, Lecture Notes in Computer Science 1807, pp. 19–34. Springer, Berlin
10. Gaudry P (in press) Index calculus for abelian varieties and the elliptic curve discrete logarithm problem. J Symbol Comput
11. Gaudry P, Thomé E, Thériault N, Diem C (2007) A double large prime variation for small genus hyperelliptic index calculus. Math Comput 76(257):475–492
12. Granger R, Hess F, Oyono R, Thériault N, Vercauteren F (2007) Ate Pairing on Hyperelliptic Curves. EUROCRYPT 2007, Lecture Notes in Computer Science 4515:430–447. Springer, Berlin
13. Menezes A, Okamoto T, Vanstone S (1993), Reducing elliptic curve logarithms in a finite field. IEEE Trans Inf Theor, IT-39(5):1639–1646
14. K. Nagao, Index calculus attack for Jacobian of hyperelliptic curves of small genus using two large primes. Jpn J Ind Appl Math 24(3):289–305
15. Pohlig S, Hellman M (1978) An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. IEEE Trans Inf Theor 24:106–110
16. Pollard J (1978) Monte Carlo methods for index computation (mod p). Math Comput 32:918–924
17. Smith B, Isogenies and the discrete logarithm problem on Jacobians of genus 3 hyperelliptic curves. Advances in Cryptology – EUROCRYPT 2008, Lecture Notes in Computer Science 4965:163–180. Springer, Berlin
18. Thériault N (2003) Weil descent attack for Kummer extensions. J Ramanujan Math Soc 18:281–312
19. Thériault N (2003) Index calculus attack for hyperelliptic curves of small genus. Advances in cryptology – ASIACRYPT 2003, Lecture Notes in Computer Science 2894:75–92. Springer, Berlin
20. Thériault N (2003) Weil descent for Artin-Schreier curves. preprint, 2003
21. Thériault N (with Avanzi R), Towards an exact cost analysis of index calculus algorithms. Presentation at ECC 2006

Hyperelliptic Curves

Roberto Avanzi[1], Nicolas Thériault[2]
[1]Ruhr-Universität Bochum, Bochum, Germany
[2]Departamento de Matemática, Universidad del Bío-Bío, Talca, Chile

Synonyms

HEC acronym is often used for hyperelliptic curves; Sometimes, however, it also stands for hyperelliptic curve cryptography. For the latter concept the acronym HECC is often used

Related Concepts

►Elliptic Curves

Definition

Similarly to ►elliptic curves, hyperelliptic curves have been suggested for cryptographic applications [4]. There are key-exchange, signature, and enciphering schemes that make use of ►groups associated to hyperelliptic curves. The rational point subgroup of the Jacobian variety of a hyperelliptic curve [1–3, 5, 6]. In some cases, the performance of these systems can rival that of ►elliptic curves.

There is also ►primality proving algorithms that make use of hyperelliptic curves by Adleman and Huang, but this method is less efficient than the elliptic curve one, and as a result is chiefly of theoretical interest.

Theory

A ►hyperelliptic curve $\mathcal{C}$ of genus g over a field $\mathbb{F}$ is defined by a Weierstraß equation of the form

$$\mathcal{C}/\mathbb{F} \; : \; y^2 + h(x)y = f(x) \tag{1}$$

such that:

1. The curve is absolutely irreducible – i.e., the polynomial $y^2 + h(x)y - f(x)$ does not factor over the algebraic closure $\bar{\mathbb{F}}$ of $\mathbb{F}$ – and is non-singular – i.e., has no singular points defined over $\bar{\mathbb{F}}$. Singular points are simultaneous solutions (x_0, y_0) to the system

$$\begin{cases} y^2 + h(x)y - f(x) = 0 \\ 2y + h(x) = 0 \\ \dfrac{dh(x)}{dx} y - \dfrac{df(x)}{dx} = 0 \end{cases}$$

   Note that the second and third equations are the partial derivatives of the curve equation with respect to the variables x, respectively y. When $h(x) = 0$ (and thus char $\mathbb{F} \neq 2$), this is equivalent to the requirement that f be square-free.
2. $\deg h(x) \leq g + 1$, and $\deg f(x) = 2g + 1$ (imaginary model) or $2g + 2$ (real model). Furthermore,
   a. In the imaginary model:
      - $f(x)$ can be taken monic
      - If char $\mathbb{F}$ is 2, $\deg h(x) \leq g$, otherwise via the change of variables $y \mapsto y - h(x)/2$, it can be assumed that $h(x) = 0$.
   b. In the real model:
      - If char $\mathbb{F}$ is odd, $f(x)$ can be made monic
      - If char $\mathbb{F}$ is even, $h(x)$ can be made monic, with $\deg h(x) = g + 1$ and either
        - $\deg f(x) = 2g + 2$ with leading coefficient $e^2 + e$ for some $e \in F^*$

or

- $\deg f(x) \leq 2g + 1$

From this definition it is clear that the ►elliptic curves are in fact hyperelliptic curves of genus one. The set of rational points of the curve $\mathcal{C}$ over a field $L \supseteq \mathbb{F}$ is defined, similarly to the case of elliptic curves, as the set

$$\mathcal{C}(L) = \{(x, y) \in L \times L : y^2 + h(x)y - f(x) = 0\} \cup \{\infty\}.$$

However, this set does not in general form a group – in fact it is a group only if the genus is one.

However, the *inverse* of a point can be defined. Let $P = (x, y)$ be a point on $\mathcal{C}$. Define $w(P) = (x, -y)$, which is also a point on $\mathcal{C}$. The map $\omega : P \mapsto \omega(P)$ is called the *hyperelliptic involution*. It satisfies $\omega(\omega(P)) = P$ for all points P on $\mathcal{C}$.

Even though protocols designed around real hyperelliptic curves have been proposed, for cryptographic applications, the imaginary model is more commonly used. The treatment in this section is therefore restricted to imaginary curves. Furthermore, only curves defined over ►finite fields are considered (although the algebraic closures of those fields also play a role).

General Curve Isomorphisms

Two hyperelliptic curves of genus g defined by Weierstrass equations

$$\mathcal{C} : y^2 + h(x)y - f(x) = 0 \quad (2)$$

$$\tilde{\mathcal{C}} : y^2 + \tilde{h}(x)y - \tilde{f}(x) = 0 \quad (3)$$

are isomorphic over a field $L \supseteq \mathbb{F}$ (which also admits the case $L = \mathbb{F}$) if there exists a map ϕ of the form

$$\phi : (x, y) \mapsto \left(s^{-2}x + b, s^{-(2g+1)}y + A(x)\right) \quad (4)$$

from the first curve to the second one, with $s \in L^{\times}$, $b \in L$ and $A(x) \in L[x]$ is a polynomial of degree at most g. A transformation of the type ϕ is called an *admissible change of variables*.

Group Law

As mentioned above, in general the set of points of a hyperelliptic curve does not form a group. To obtain a group from a hyperelliptic curve $\mathcal{C}$, it is necessary to consider its *divisor class group* $\mathrm{Pic}^0_{\mathcal{C}}(L)$ instead, which is isomorphic to the L-rational point subgroup of Jacobian variety of the curve $\mathcal{C}$.

The elements of the divisor class group are quotient classes of formal sums of points on the curve – these sums are called *divisors*.

The divisor class group $\mathrm{Pic}^0_{\mathcal{C}}(L)$ of $\mathcal{C}$ over L of degree zero is the quotient of the group of degree zero divisors $\mathrm{Div}^0_{\mathcal{C}}(C)$ by the principal divisors. It is also called the Picard group of $\mathcal{C}$.

The degree zero divisors on the curve, i.e., formal sums $\sum_{P \in \mathcal{C}(\bar{L})} m_P P$ with almost but a finite amount of the coefficients vanishing and the sum $\sum_{P \in \mathcal{C}(\bar{L})} m_P = 0$. A divisor D is L-rational if it is invariant under the action of automorphisms of $\mathrm{Gal}(\bar{L}/L)$ – in other words the points with nonzero coefficients are not necessarily all defined over L but if one is present, the whole Galois orbit containing it must be in the divisor and with the same coefficients.

The principal divisors are the divisors associated to L-rational functions on the curve. That is, if $r(x, y)$ is a rational function on $\mathcal{C}$ and P is a point on $\mathcal{C}(\bar{L})$, then m_P is defined as zero if the function $r(x, y)$ does not vanish nor has a pole at P; is equal to the order of the zero if $r(x, y)$ vanishes at P; and is equal to minus the order of the pole otherwise. Under this definition, $\sum_{P \in \mathcal{C}(\bar{L})} m_P P$ is the principal divisor associated to $r(x, y)$.

Mumford introduced a representation of the elements of the divisor class group as polynomial pairs, for which Cantor provided an explicit arithmetic algorithm. Mumford's representation is a one-to-one correspondence between the elements of the divisor class group and the pairs $[U(t), V(t)]$ of polynomials over L satisfying:

1. $U(t)$ is monic.
2. $\deg V(t) < \deg U(t) \leq g$.
3. $V(t)^2 + V(t)h(t) - f(t)$ is a multiple of $U(t)$.

At the core of this correspondence, there is the following identification of such a pair with a divisor: If $U(t) = \prod_{i=1^k}(t - x_i)^{m_i}$ with the x_i pairwise distinct, then the pair $[U, V]$ is associated to the divisor having the points $(x_1, V(x_1)), (x_2, V(x_2)), \ldots, (x_k, V(x_k))$ on its finite support, with multiplicities $m_1, m_2, \ldots, m_k$ respectively. Note that if a point P appears in the support of such a divisor, then its inverse $\omega(P)$ does not, and all multiplicities are nonnegative. The pair $[U(t), V(t)]$ is said to represent the *semi-reduced divisor D*. A *reduced divisor* is a semi-reduced divisor that additionally satisfies $\deg U(t) \leq g$. In each divisor class there always exists a unique reduced element.

Cantor's algorithm is given as Algorithm 1. Note, however, that the algorithm is seldom used in this form. Instead, several optimizations are usually performed. For instance, the field operations implicit in all the polynomial operations are made explicit, where one considers usually only the most common case (where the polynomials U_i and V_i are of maximal degree and have no common roots), and all redundant operations (computations of coefficients that have no impact on the final result or duplicate operations) are not performed. There is an extensive, and still growing, literature on these explicit formulas for hyperelliptic curves, a field that was initiated by Robert Harley.

Group Order and Structure

Let $\mathcal{C}/\mathbb{F}_q$ be a hyperelliptic curve of genus g defined over the finite field with q elements. The structure of the $\mathbb{F}_q$-rational divisor class group is of the form

$$\mathrm{Pic}^0_{\mathcal{C}}(\mathbb{F}_q) \simeq \frac{\mathbb{Z}}{n_1\mathbb{Z}} \times \frac{\mathbb{Z}}{n_2\mathbb{Z}} \times \cdots \times \frac{\mathbb{Z}}{n_{2g}\mathbb{Z}},$$

where $n_i \mid n_{i+1}$ for $1 \le i < 2g$ and for all $1 \le i \le g$ one has $n_i \mid q-1$.

By the Hasse–Weil Theorem, it is easy to obtain bounds on the number of points on the curve and its Jacobian, called Hasse–Weil bounds as for elliptic curves:

1. $\left| |C(\mathbb{F}_q)| - q - 1 \right| \le 2g\sqrt{q}$.
2. $\left| |\mathrm{Pic}^0_C| - q^g \right| = O\left(q^{g-1/2}\right)$.

Serre has given a sharper estimate for the first bound: $||C(\mathbb{F}_q)| - q - 1| \le g\lfloor 2\sqrt{q} \rfloor$.

Examples

First Example

Consider the curve

$$\mathcal{C} : y^2 = x^5 + 4x^3 + 3x^2 + 4x - 4$$

over $\mathbb{F}_{11}$. It is easy to verify that this curve is non-singular (the polynomial $x^5 + 4x^3 + 3x^2 + 4x - 4$ is a square-free). The 15 points on this curve are

$$\infty, (2,5), (2,6), (3,1), (3,10), (4,3), (4,8), (5,3),$$
$$(5,8), (6,1), (6,10), (8,1), (8,10), (10,1), (10,10).$$

The cardinality of divisor class group $\mathrm{Pic}^0_{\mathcal{C}}(\mathbb{F}_{11})$ is 163. This is a ►prime number. Hence, all group elements except the neutral element are generators of the group. One of them is given as

$$D := [\,t^2 - t - 2\,,\,5t - 5\,]$$

in Mumford's representation. The first polynomial factors as $(t-10)(t-2)$, so 10 and 2 are the x-coordinates of two points on the curve. Evaluating the second polynomial at $x = 10$ and $x = 2$ gives 1 and 5. Indeed $(10,1)$ and $(2,5)$ are points on $\mathcal{C}$.

Algorithm 1. Cantor's operation for hyperelliptic divisor class groups

INPUT: Reduced divisors $D_1 = [U_1(t), V_1(t)]$ and $D_2 = [U_2(t), V_2(t)]$
OUTPUT: Reduced divisor $D_3 = [U_3(t), V_3(t)]$, $D_3 = D_1 + D_2$

```
1.  Composition: $[U_C(t), V_C(t)] = D_1 + D_2$ (semi-reduced)
2.      Write $d(t) = \gcd(U_1, U_2, V_1 + V_2 + h) = r_1U_1 + r_2U_2 + r_3(V_1 + V_2 + h)$
                                                   [Extended Euclidean Algorithm]
3.    $\tilde{U}(t) \leftarrow U_1U_2/d^2$
4.    $\tilde{V}(t) \leftarrow V_2 + \frac{U_2}{d} r_2(V_1 + V_2) + r_3\frac{V_2^2 + hV_2 + f}{d} \bmod U_C$
5.  Reduction: $D_3 = [U_3(t), V_3(t)]$ (reduced)
6.      while $\deg \tilde{U}(t) > g$ do
7.          $\tilde{U}(t) \leftarrow Monic\left(\frac{\tilde{V}^2 + h\tilde{V} + f}{\tilde{U}}\right)$
8.          $\tilde{V}(t) \leftarrow \tilde{V} + h \bmod \tilde{U}$
9.      $U_3(t) \leftarrow \tilde{U}$, $V_3(t) \leftarrow \tilde{V}$
10. return $[U_3(t), V_3(t)]$
```

On the other hand, one can compute (using Cantor's algorithm for instance), that $5 \cdot D$ is the divisor

$$[\, t^2 - x - 4\,,\ -5\,t - 2\,],$$

where the first polynomial is irreducible. This means that the finite support of this divisor consists of two points defined over $\mathbb{F}_{11^2}$, which are Galois conjugates. Yet, the divisor itself is $\mathbb{F}_{11}$-rational.

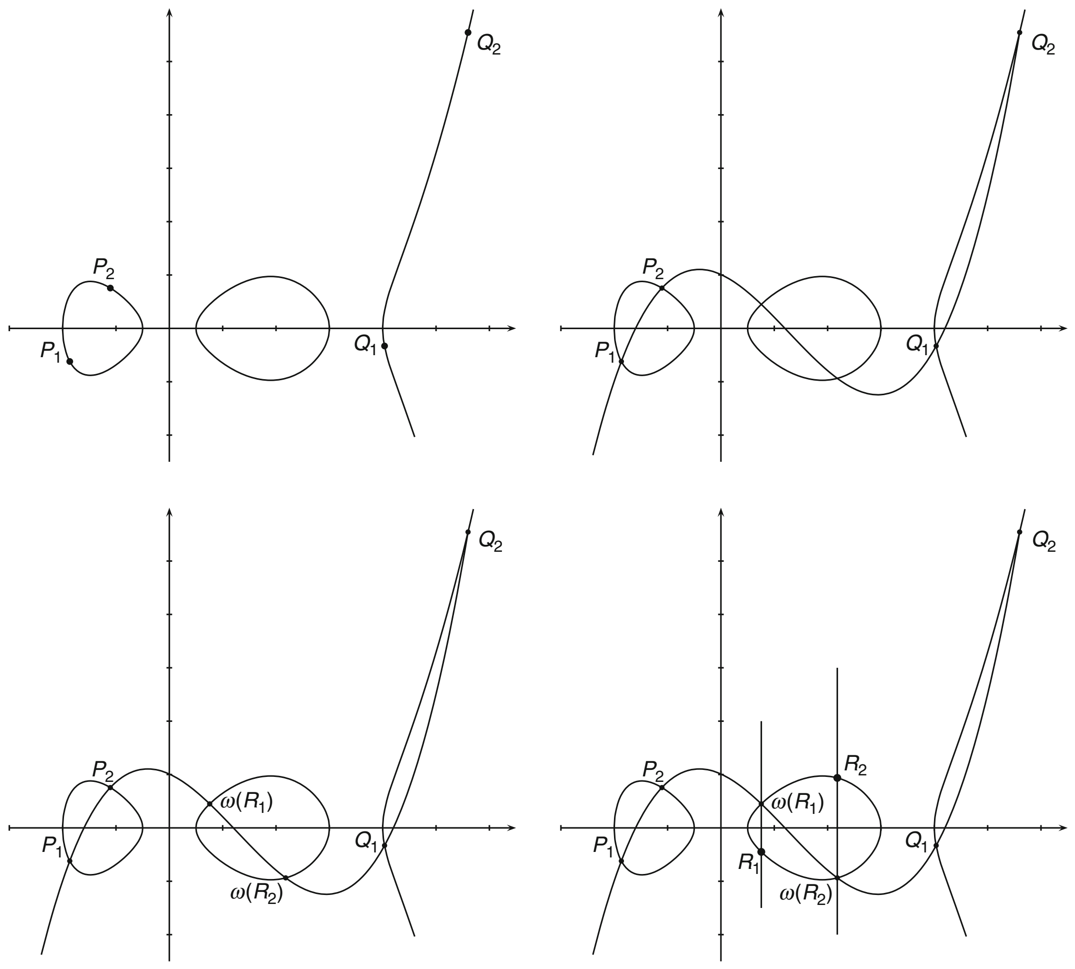
Second Example

It is also possible to give a graphical example of divisor composition and reduction: Consider the following genus two curve (over the field of real numbers):

$$\mathcal{C}\ :\ y^2 = x^5 - 5\,x^4 - \frac{9}{4}\,x^3 + \frac{101}{4}\,x^2 + \frac{1}{2}\,x - 6.$$

Suppose that one needs to add the two reduced divisors $D_1 = P_1 + P_2 - 2\infty$ and $D_2 = Q_1 + Q_2 - 2\infty$ on $\mathcal{C}$. The steps of Cantor's algorithm are depicted in Fig. 1. The composition is easy: simply collect all the points together, giving the divisor $\tilde{D} := D_1 + D_2 = P_1 + P_2 + Q_1 + Q_2 - 4\infty$.

For the reduction, it is required to add (or subtract) principal divisors, i.e., divisors of rational functions. One such function is the only cubic that passes through the four points P_1, P_2, Q_1, and Q_2. This curve intersects the hyperelliptic curve in two further points, which can be called $\omega(R_1)$ and $\omega(R_2)$, where R_1 and R_2 are two other points on the curve. Since $P_1 + P_2 + Q_1 + Q_2 + \omega(R_1) + \omega(R_2) - 6\infty$ is a principal divisor, it is equivalent to the zero divisor in the divisor class group, and thus $\omega(R_1) + \omega(R_2) - 2\infty$ must be the opposite of $\tilde{D}$. In order to invert $\omega(R_1) + \omega(R_2) - 2\infty$, observe that for any point P, the divisor $P + \omega(P) - 2\infty$ is the principal divisor corresponding to the equation of a vertical line, and thus the inverse of $\omega(R_1) + \omega(R_2) - 2\infty$ is $R_1 + R_2 - 2\infty$, which gives the reduced divisor corresponding to the sum of D_1 and D_2.



Hyperelliptic Curves. Fig. 1 Addition of two reduced divisors on an hyperelliptic curve

Recommended Reading

1. Avanzi R, Cohen H, Frey G, Lange T, Nguyen K, Vercauteren F (2006) Handbook of elliptic and hyperelliptic curve cryptography. CRC Press, Boca Raton
2. Blake IF, Seroussi G, Smart NP (2005) Advances in elliptic curve cryptography. London Mathematical Society Lecture Note Series 317, Cambridge University Press, Cambridge
3. Jacobson M Jr, Menezes A, Stein A (2004) Hyperelliptic curves and cryptography. In: High Primes and Misdemeanors: Lectures in Honour of the 60th Birthday of Hugh Cowie Williams, Fields Institute Communications 41, American Mathematical Society, pp. 255–282, Providence
4. Koblitz N (1989) Hyperelliptic cryptosystems. J Cryptol 1:139–150
5. Menezes A, Wu Y, Zuccherato R (1998) An elementary introduction to hyperelliptic curves. In Algebraic aspects of cryptography, Springer, New York
6. Washington L (2008) Elliptic curves: number theory and cryptography, 2nd edn. CRC Press, Boca Raton

Hyperelliptic Curves Performance

Roberto Avanzi[1], Nicolas Thériault[2]
[1]Ruhr-Universität Bochum, Bochum, Germany
[2]Departamento de Matemática, Universidad del Bío-Bío, Talca, Chile

Synonyms

Efficiency of hyperelliptic curve cryptosystems

Related Concepts

►Elliptic Curve Cryptography; ►Elliptic Curves; ►Hyperelliptic Curves

Definition

Evaluating the performance of hyperelliptic curve cryptosystems is done by comparing the time required to do the encryption compared with other cryptosystems – mainly those based on ►elliptic curves – providing they offer the same level of security.

Background

Only 4 years after suggesting to use of ►elliptic curves for discrete logarithm-based cryptographic applications, Koblitz [14] proposed to use the Jacobian variety of hyperelliptic curves for the same purpose. This is a natural generalization since elliptic curves are in fact hyperelliptic curves of genus one.

Hyperelliptic curves (of genus larger than one) have long been thought not to be competitive with elliptic curves because of some outstanding issues. It is more difficult to count points on the Jacobian of a hyperelliptic curve than on an elliptic curve. Explicit construction methods of varieties with rational point groups of a given order are better understood for elliptic curves than for higher genus varieties. The ►group operations were expected to be much slower than for elliptic curves, to the point that it was expected hyperelliptic curves would offer no performance advantage over elliptic curves [20].

However, the situation has considerably changed in the last few years. ►Point counting algorithms can now determine the cardinality of genus two curves of cryptographically good order in reasonable time: see for instance [12], and http://www.loria.fr/~gaudry/record127/ for a recent record. For curves defined over prime fields, it is possible to count points in curves of genus two, or use the complex multiplication (CM) method for genus two [17, 22] and three [22]. All these methods have been improved considerably since their publication. For the genus two case, Streng [21] and Satoh [19] provide the best algorithms to date.

Application and Experimental Results

The performance of the group operation has also been considerably improved. In many cases it matches that of elliptic curves, and in other cases it even surpasses it. The purpose of this essay is to review some recent performance results that show the competitiveness of hyperelliptic curves.

Curves over Prime Fields

The classical method to perform explicit computations in divisor class groups of hyperelliptic curves of genus at least two, namely Cantor's algorithm [7] is very slow compared to elliptic curve group operations, including the most basic chord-and-tangent formulas in affine coordinates.

The first efficient explicit formulas for genus two curves, due to Harley [13], were followed by very active research on the explicit arithmetic of low genus curves. Several optimization techniques were introduced, notably by Nagao [18], reducing the number of field operations needed to implement a group operation, culminating in the complete set genus two formulas by Lange [15]. Lange was able to give fast formulas for hyperelliptic curves in affine coordinates using only one inversion, and then the first instances of two different types of inversion-free weighted coordinates. As was first showed by Avanzi [1], using careful implementation of finite field arithmetic and Lange's formulas it is possible to get a performance for genus two curves which is very close to that of elliptic curves.

Using theta functions and a Montgomery ladder, Gaudry [10] derived fast formulas for the scalar multiplication in the Kummer surface associated to a genus two curve. He described a scalar multiplication requiring only 25 field multiplications per scalar bit.

In practice, a genus two hyperelliptic curve must be defined on a field with exactly half the size (in bits) of the field of definition of an elliptic curve if they are to provide the same security level. From this observation, Gaudry's formulas would be expected to yield performances similar to that of elliptic curve scalar multiplication methods costing roughly 6.25 field multiplications per scalar bit. As pointed out by Bernstein [8, Part I], such an analysis is in fact flawed, but since 6 of these 25 multiplications are done with constants coming from the curve equation, choosing the right curve parameters allows Gaudry's method to outperform Montgomery ladders for elliptic curves.

The introduction of a new model for elliptic curves by Edwards [11] made the competition between genus one and higher genus curves more interesting. Bernstein and Lange investigated these curves in depth and, along with a number of other authors, devised explicit formulas for Edwards curves. These formulas offer slightly faster addition and doubling than the best previously known methods for elliptic curves. These gains allowed elliptic curves to be again faster than genus two curves. However, the performance differences are still relatively contained, and the last word in this race remains to be written.

Curves over Binary Fields

Just as Cantor's algorithm could be adapted to the case of hyperelliptic curves over fields of characteristic two by Koblitz, explicit formulas for divisor addition and doubling were also devised for the even characteristic case.

The impact of characteristic two on divisor doubling is particularly interesting. Since many operations in Cantor's algorithm involve the squaring of polynomials, these parts of the algorithm become particularly efficient over binary fields. A striking example of the performance of doubling over genus two curves if suitable curve parameters are chosen is given in [16]. For an overview of genus two coordinate systems and formulas see [15].

Several authors [4, 6, 9] also adapted halving methods from the elliptic curve setting to use them for genus two and three curves. In many cases, divisor halvings can be significantly faster than divisor doublings, thus significantly improving the performance of genus two and three scalar multiplication on these curves.

Avanzi, Thériault, and Wang [3] give an explicit curve arithmetic for genera up to four, with specific improvements for genus three and four curves. In turn, this is based on a careful implementation of finite field of characteristic two for each required field size [2]. A comparison between curves of genus up to four was done, with each curve using the best doubling-based coordinate system available. The conclusion of this comparison is that curves of genus two and three have similar performance, even after taking into account the increase in field sizes needed because of index-calculus attacks on genus three hyperelliptic curves, and they are both faster than elliptic curve (often by as much as 50%). In fact, in some cases genus three curves are the fastest – this depending mainly on the relation between finite field sizes in relation to the granularity of the target architecture of the implementation. In the same paper, the performance of the group arithmetic for curves of genus four is seen to be comparable to that of elliptic curves. As with curves over prime fields, the last word has yet to be said about the performance of hyperelliptic curves over binary fields, but it can be safely said that low genus hyperelliptic curves are a sound alternative to elliptic curves.

Recommended Reading

1. Avanzi R (2004) Aspects of hyperelliptic curves over large prime fields in software implementations. Cryptographic Hardware and Embedded Systems – CHES 2004, Lecture Notes in Computer Science 3156, pp. 148–162. Springer, Berlin
2. Avanzi R, Thériault N (2007) Effects of optimizations for software implementations of small binary field arithmetic. Arithmetic of Finite Fields – WAIFI 2007, Lecture Notes in Computer Science 4547, pp. 69–84. Springer, Berlin
3. Avanzi R, Thériault N, Wang Z (2008) Rethinking low genus hyperelliptic Jacobian arithmetic over binary fields: interplay of field arithmetic and explicit formulæ. J Math Cryptol 2:227–255
4. Birkner P (2007) Efficient divisor class halving on genus two curves. Selected Areas in Cryptography SAC 2006, Lecture Notes in Computer Science 4356, pp. 317–326. Springer, Berlin
5. Birkner P, Thériault N (2008) Faster halvings in genus 2. Selected Areas in Cryptography SAC 2008, Lecture Notes in Computer Science 5381, pp. 116. Springer, Berlin
6. Birkner P, Thériault N (2009) Efficient halvings for genus 3 curves over binary fields. preprint, 2009
7. Cantor DG (1987) Computing in the Jacobian of hyperelliptic curves. Math Computation 48:95–101
8. Bernstein DJ, Lange T. Series of talks on Elliptic vs. hyperelliptic
9. Fan X, Wollinger TJ, Wang Y (2006) Efficient doubling on genus 3 curves over binary fields. Topics in Cryptology CT-RSA 2006, Lecture Notes in Computer Science 3860, pp. 64–81. Springer, Berlin
10. Gaudry P (2007) Fast genus 2 arithmetic based on Theta functions. J Math Cryptol 1:243–265
11. Edwards HM (2007) A normal form for elliptic curves. Bull Am Math Soc (N.S.) 44(3):393–422
12. Gaudry P, Schost É (2004) Construction of Secure Random Curves of Genus 2 over Prime Fields. Advances in Cryptology – EUROCRYPT 2004, Lecture Notes in Computer Science 3027, pp. 239–256. Springer-Verlag, Berlin
13. Harley R (2000) Fast arithmetic on genus two curves. Preprint
14. Koblitz N (1989) Hyperelliptic cryptosystems. J Cryptol 1:139–150

15. Lange T (2005) Formulae for arithmetic on genus 2 hyperelliptic curves. Applicable Algebra in Engineering, Commun Comput 15:295–328
16. Lange T, Stevens M (2005) Efficient doubling for genus two curves over binary fields. Selected Areas in Cryptography – SAC 2004, Lecture Notes in Computer Science 3357, pp. 170–181. Springer, Berlin
17. Mestre JF (1991) Construction des courbes de genre 2 a partir de leurs modules. Prog Math 94:313–334
18. Nagao K (2000) Improving group law algorithms for Jacobians of hyperelliptic curves. Algorithmic Number Theory – ANTS-IV, Lecture Notes in Computer Science 1838, pp. 439–448. Springer, Berlin
19. Satoh T (2009) Generating genus two hyperelliptic curves over large characteristic finite fields. Advances in Cryptology – EUROCRYPT 2009, Lecture Notes in Computer Science 5479, pp. 536–553. Springer, Berlin
20. Smart N (1999) On the performance of hyperelliptic cryptosystems. Advances in Cryptology – EUROCRYPT '99, Lecture Notes in Computer Science 1592, pp. 165–175. Springer, Berlin
21. Streng M (2008) Computing Igusa Class Polynomials. Preprint
22. Weng A (2001) Konstruktion kryptographisch geeigneter Kurven mit komplexer Multiplikation. PhD thesis, Universität Gesamthochschule Essen, Germany