F

Face Recognition from Still Images and Video

Soma Biswas[1], Rama Chellappa[2]
[1]Department of Electrical and Computer Engineering, Center for Automation Research, University of Maryland, College Park, MD, USA
[2]Center for Automation Research, Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA

Related Concepts

►Biometric Identification; ►Verification

Definition

Face recognition is concerned with identifying or verifying one or more persons from still images or video sequences using a stored database of faces.

Background

The earliest work on face recognition started as early as 1950s in psychology and in the 1960s in engineering, but research on automatic face recognition practically started in the 1970s after the seminal work of Kanade [1] and Kelly [2].

Application

Face recognition has wide range of applications in many different areas ranging from law enforcement and surveillance, information security to human-computer interaction, virtual reality, and computer entertainment.

Introduction

Face recognition with its wide range of commercial and law enforcement applications has been one of the most active areas of research in the field of computer vision and pattern recognition. Personal identification systems based on faces have the advantage that facial images can be obtained from a distance without requiring cooperation of the subject, as compared to other biometrics such as fingerprint, iris, etc. Face recognition is concerned with identifying or verifying one or more persons from still images or video sequences using a stored database of faces. Depending on the particular application, there can be different scenarios, ranging from controlled still images to uncontrolled videos. Since face recognition is essentially the problem of recognizing a 3D object from its 2D image or a video sequence, it has to deal with significant appearance changes due to illumination and pose variations. Current algorithms perform well in controlled scenarios, but their performance is far from satisfactory in uncontrolled scenarios. Most of the current research in this area is focused toward recognizing faces in uncontrolled scenarios. This chapter is broadly divided into two sections. The first section discusses the approaches proposed for recognizing faces from still images and the second section deals with face recognition from video sequences.

Still Image Face Recognition

This section discusses some of the early subspace and feature-based approaches, followed by those which address the problem of appearance change due to illumination variations and approaches that can handle both illumination and pose variations.

Early Approaches

Among the early subspace-based holistic approaches, eigenfaces [3] and Fisherfaces [4, 5] have proved to be very effective for the task of face recognition. Since human faces have similar overall configuration, the facial images can be described by a relatively low-dimensional subspace. Principal component analysis (PCA) [3] has been used for finding those vectors which can best account for the distribution of facial images within the whole image space. These vectors are eigenvectors of the covariance matrix computed from the aligned face images in the training set and are thus known as "eigenfaces." Given the eigenfaces, every face in the gallery database is represented as a vector of weights obtained by projecting the image onto the eigenfaces using inner product operation. The identification of a new test image is done by locating

Henk C.A. van Tilborg & Sushil Jajodia (eds.), *Encyclopedia of Cryptography and Security*, DOI 10.1007/978-1-4419-5906-5,

the image in the database whose weights are the closest to the weights obtained for the test image, by projecting onto the eigenfaces.

In contrast to PCA, linear discriminant analysis (LDA)/ Fisher linear discriminant (FLD) [4][5] utilizes the available class information for performing face recognition. LDA aims at finding an optimal projection of the labeled data such that the ratio of the between-class scatter and within-class scatter is maximized. For a C-class problem, where each class corresponds to one subject, the between and within-class scatter matrices S_w and S_b are given by

$$S_b = \sum_{i=1}^{C} N_i(\boldsymbol{u}_i - \boldsymbol{u})(\boldsymbol{u}_i - \boldsymbol{u})^T; \; S_w = \sum_{i=1}^{C} \sum_{\boldsymbol{x}_k \in Xi} (\boldsymbol{x}_k - \boldsymbol{u}_i)(\boldsymbol{x}_k - \boldsymbol{u}_i)^T \tag{1}$$

Here, $\boldsymbol{u}_i$ denotes the mean image of the ith class X_i containing N_i number of samples and $\boldsymbol{u}$ is the mean of all the classes. The optimal projection matrix W, which maximizes $|W^T S_b W|/|W^T S_w W|$ and the optimal projections vectors can be solved using the following generalized eigenvalue problem: $S_b w_i = \lambda_i S_w w_i$, where λ_i are the corresponding eigenvalues. For classification, the test image is first projected onto the subspace spanned by the computed projection vectors. The projection coefficients are then compared to all the stored vectors of labeled classes to determine the input class label.

In addition to these subspace-based methods, feature-based graph matching approaches have also been quite successful for face recognition. Compared to the holistic approaches, feature-based methods are less sensitive to variations in illumination, viewpoint, face localization accuracy, etc., but depend on reliable and accurate feature extraction techniques. One of the most successful methods in this category is the elastic bunch graph matching [6]. In this approach, each facial landmark (like the pupils, mouth corners, nose tip, etc) is associated with a local feature representation called "jet" which consists of wavelet coefficients at different scales and rotations. A face is represented as a labeled graph, with the nodes located at these facial landmarks. To account for the wide range of variations in the facial appearance, the representation is made richer by attaching a representative set of jets to each landmark (called the face bunch graph representation). For a given test image, the corresponding image graph is compared to all model graphs and the one having the highest similarity is selected.

Face Recognition Across Illumination Variations

The appearance of a facial image changes considerably with variations in illumination conditions. Jacobs et al. [7] showed that given two images under different illumination conditions, there always exists a family of physically realizable objects which is consistent with both images. And this ambiguity exists even under the hard constraints of Lambertian reflectance and known light source directions. Suppose I and J be two images of different Lambertian objects, illuminated by two different known distant light sources, $l = (l_x, l_y, l_z)$ and $k = (k_x, k_y, k_z)$, respectively. If p_x and q_y denote the surface gradients and $\rho(x, y)$ denotes the unknown albedo map, from the standard image irradiance equation,

$$I(x,y) = \rho(x,y) \frac{-(l_x, l_y, l_z) \cdot (p_x, q_y, 1)}{\sqrt{p_x^2 + q_y^2 + 1}}$$
$$J(x,y) = \rho(x,y) \frac{-(k_x, k_y, k_z) \cdot (p_x, q_y, 1)}{\sqrt{p_x^2 + q_y^2 + 1}} \tag{2}$$

It can be proved that given any arbitrary pair of images, there exists a set of albedo and surface gradients that satisfies both the above equations simultaneously [7] and thus, theoretically, it is not possible to determine if the two images are from the same object or not. But this analysis does not impose any constraint on the class of objects that is being considered. So though illumination-invariant image matching is an ill-posed problem, for matching faces across illumination variations, different face-specific information can be utilized to make the problem more tractable.

Assuming that faces belong to the class of linear Lambertian objects, Zhao et al. [8] proposed an approach that is robust to illumination variations. For Lambertian objects, the diffused component of the reflectance is given by Lambert's law which relates the pixel intensity to the albedo, shape of the surface point and the illumination direction as follows

$$h_i = (\rho n^T)_i s = t_i^T s \tag{3}$$

By stacking the intensities of all the pixels, an image with d pixels can be expressed as

$$\boldsymbol{h}_{d\times1} = [h_1, h_2, ..., h_d]^T = \boldsymbol{T}_{d\times3} \cdot \boldsymbol{s}_{3\times1} \tag{4}$$

Here the matrix $\boldsymbol{T}$ encodes the product of albedos and surface normals for all the pixels and is known as the object-specific albedo-shape matrix. Further, the linear property leads to a rank constraint on the shape-albedo matrix. So if $\boldsymbol{T}$ can be represented as a linear combination of m basis $\boldsymbol{T}_i$'s, that is $\boldsymbol{T} = f_1\boldsymbol{T}_1 + f_2\boldsymbol{T}_2 + \cdots + f_m\boldsymbol{T}_m$, the image can be expressed as

$$\boldsymbol{h}_{d\times1} = [\boldsymbol{T}_1, \boldsymbol{T}_2, \cdots, \boldsymbol{T}_m](\boldsymbol{f} \times \boldsymbol{I}_3).\boldsymbol{s} = \boldsymbol{W}_{d\times3m}[\boldsymbol{f}_{m\times1} \times \boldsymbol{s}_{3\times1}] \tag{5}$$

Here, the matrix $\boldsymbol{W}$ encodes the albedo maps and surface normals for a class of objects and is called the class-specific albedo-shape matrix. Given the input image, the combining coefficients $\boldsymbol{f}$ and the illumination $\boldsymbol{s}$ can be obtained in an iterative manner. The coefficients $\boldsymbol{f}$, which can be thought of as the illumination-free identity vector, can be used for face recognition across illumination variations.

Any real world object is characterized by its underlying shape and surface properties. Unlike image intensity, these intrinsic characteristics of an object are insensitive to changes in illumination conditions which make them useful for matching faces across illumination variations. In a recent approach [9], an image estimation framework has been proposed for robust estimation of the facial albedo for face recognition application. The basic intuition is that if $\boldsymbol{n}_{i,j}^{(0)}$ and $\boldsymbol{s}^{(0)}$ be some initial values of the surface normal and illuminant direction, the Lambertian assumption imposes the following constraint on the initial albedo $\boldsymbol{\rho}^{(0)}$ obtained

$$\boldsymbol{\rho}_{i,j}^{(0)} = \frac{\boldsymbol{I}_{i,j}}{\boldsymbol{n}_{i,j}^{(0)} \cdot \boldsymbol{s}^{(0)}} \tag{6}$$

Here, $\boldsymbol{n}^{(0)}$ can be the average facial surface normal computed from a collection of 3D facial data. Clearly, the inaccurate initial estimates of normals and light source direction result in errors in the initial albedo map. It can be shown [9] that this initial erroneous albedo map can be written as a sum of the true albedo and a signal-dependent additive noise. This is a typical image estimation formulation and standard methods can be used to solve for the true unknown albedo map which can be used as an illumination-insensitive signature for face recognition across illumination variations. Figure 1 shows the albedo maps obtained from several images of a subject from the PIE dataset taken under different illumination conditions. The final albedo estimates do not seem to have the illumination effects present in the input images and appear quite similar to each other as desired. The estimated albedo maps can further be used to normalize the input images for recovering the shapes and to generate novel views of the subject under novel illumination conditions (Fig. 2).

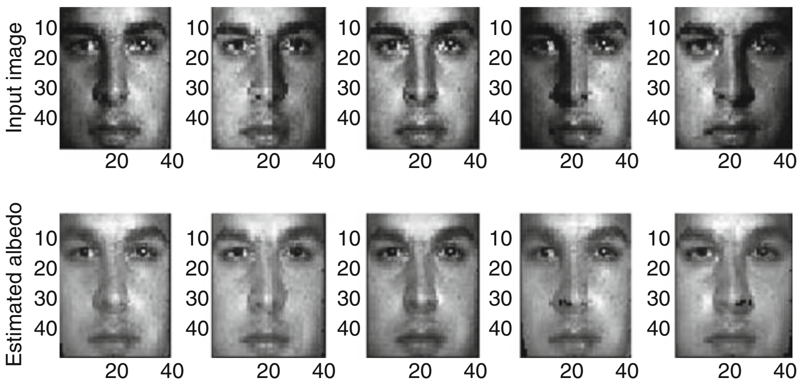
Face Recognition Across Illumination and Pose Variations

The wide variations in facial appearance due to the combined effects of illumination and pose variations can be accounted for by simulating the process of image formation. The main idea is to relate the input image with the actual 3D shape, albedo map, projection model, and incident illumination. Using a 3D morphable model, Blanz and Vetter proposed such an approach [10] to estimate the shape and texture from a single face image, which can then be used for face recognition. The morphable face model is constructed such that the shape $\mathbf{S}$ and texture map $\mathbf{T}$ of a given face can be approximated by a linear combination of shape and texture vectors $\mathbf{S}_i$ and $\mathbf{T}_i$ of a set of 3D faces

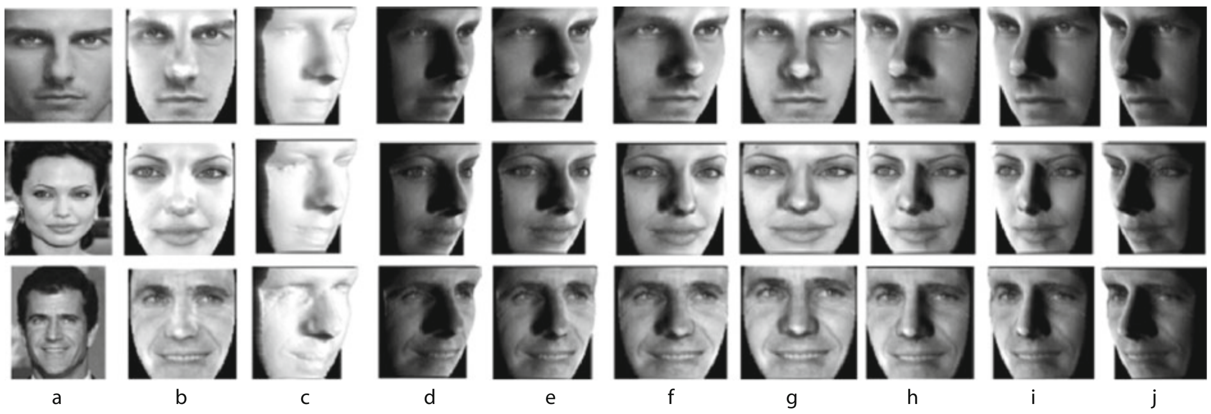
$$\mathbf{S} = \sum_{i=1}^{N} \alpha_i \mathbf{S}_i; \qquad \text{and} \qquad \mathbf{T} = \sum_{i=1}^{N} \beta_i \mathbf{T}_i \tag{7}$$

Rendering an image involves mapping the 3D surface points to the image plane coordinates using a projection model, and then determining the pixel intensities based on the illumination model and light sources. Given a test image, the shape and texture coefficients are determined along with the other unknown parameters like pose angle, source direction, etc., so as to minimize the difference between the true image and the reconstructed image. These parameters are reasonably stable across illumination and pose variations and so can be utilized to determine the identity of the individual.

Another way to address the problem of face recognition across illumination and pose variations is to



Face Recognition from Still Images and Video. Fig. 1 Albedo estimates (second row) obtained for several images of a subject (first row) from the PIE dataset (courtesy Biswas et al. [9])



Face Recognition from Still Images and Video. Fig. 2 Novel view synthesis using estimated albedo maps and shape (**a**) Input image (**b**) Estimated albedo (**c**) Recovered shape (**d–j**) Synthesized views under novel illumination conditions (Courtesy Biswas et al. [9])

characterize the set of images of an object under all possible combinations of pose and illumination. Recent results have showed that an image of a Lambertian object taken under any arbitrary illumination conditions can be approximately represented as a linear combination of the first nine spherical harmonic basis images [11, 12]. Given an input image, if these spherical harmonic basis images can be estimated, then the database face for which there exists a weighted combination of basis images that is the closest to the test face image determines the identity of the test image. Zhang and Samaras [13] proposed an approach for estimating these basis images from a single input image. First, a bootstrap set of basis images is used to build a statistical model for each of the spherical harmonic basis images. The statistical models aim to learn the probability density function for the basis images assuming Gaussian distributions of unknown means and covariances. Given an input image, these learned statistical models are used to compute the maximum a posteriori (MAP) estimate of the basis images that span the illumination space for the input subject. But in this approach, since the basis images are built from 2D images, the statistical model need to be rebuilt for each pose for handling different poses. To overcome this limitation, the statistical models can be built directly in 3D spaces by combining the spherical harmonic illumination representation and the 3D morphable model to recover basis images across both pose and illumination variations.

The spherical harmonics representation can also be used to encode pose information, in addition to modeling illumination variations [14]. The property that rotations of spherical harmonic of any order can be linearly composed entirely of other spherical harmonics of the same order can be used to derive close-form linear transformations relating the 2D harmonic basis images at different poses. So from just one set of basis images at a fixed pose, the basis images at any other pose can be synthesized analytically. These basis images can in turn be used to synthesize novel views of the subject under arbitrary lighting conditions. Given frontal gallery images for each subject, first the basis images are recovered using a statistical model [13]. For testing, the test image which can be under any pose and illumination is wrapped to a frontal view using manually established image correspondence between the test image and a mean frontal face image. This synthesized frontal image can then be directly projected on the existing frontal view basis images of the gallery subjects. The gallery face for which there exists a linear combination of basis images that is the closest to the input test face image is taken to be the identity of the test image. Novel images of the chosen subject at the same pose as the test image can also be synthesized using the closed form linear transformation between the harmonic basis images and then compared with the test image for further validation. The pose of the test image is estimated from a few manually selected facial features.

In addition to these model-based approaches, another completely different approach based on soft attributes is gaining much popularity because of its robustness to variations in pose, illumination, expression, and other imaging conditions and impressive performance for large-scale face recognition application [15]. The main idea is that a facial image can be described as, say, an *Asian old male* or a *Caucasian girl*. Based on this intuition, different binary classifiers can be trained to recognize the presence or absence of describable aspects of visual appearance or attributes like gender, race, age, etc. For each face image, different low-level features like image intensities, edge magnitudes, gradient directions, etc., are extracted for different facial regions like eyes, nose, mouth, etc. This produces a large number of possible low-level features, a subset of which is automatically chosen and used for each attribute classifier. Sometimes, an image may be more easily described in terms of the similarity of different parts of their face to a limited set of *reference* subjects. For example, the eyes may be similar to those of one person, whereas the nose may be similar to some other person. In such cases, simile classifiers can be trained, which learns the similarity of faces, or regions of faces, to specific reference people. This approach has an added advantage of not requiring the manual labeling required for attribute classification. Two faces can then be compared by comparing their trait vectors. Both the approaches give very good performance on real-world images exhibiting significant variations in pose, expression, hair-style, etc., without requiring any costly alignment between image pairs.

Video-based Face Recognition

Though face recognition research has traditionally concentrated on recognition from still images, recently video-based face recognition has also gained a lot of attention. Faces are essentially articulating 3D objects. For faces, motion encodes far richer information than simply the 3D structure of the face in the form of behavioral cues such as idiosyncratic head movements and gestures, which can potentially aid in recognition tasks. The video sequence can also be utilized to obtain more effective representations like 3D face models or super-resolution images for improving the recognition results. In fact, psychophysical studies [16] have shown that humans tend to rely more on dynamics than the structure under nonoptimal viewing conditions such as low spatial resolution, harsh illumination conditions, etc.

One of the simplest ways to use the extra information present in a video sequence as compared to still images is to fuse the results obtained by a 2D face classifier for each frame of the sequence. Here, the video sequence is viewed as an unordered set of images for both training and testing. During testing, each frame of the test video sequence independently votes for a particular identity for the test subject. Appropriate fusion techniques can then be applied to provide the final identity. Fusion can be performed at the score level where the matching scores obtained at each frame are combined to get the final score, or at the decision level where frame-level decisions are combined into a final decision. These approaches do not take advantage of the temporal information that is also present in the video sequence.

Several methods have been proposed for extracting more descriptive appearance models from videos for recognition. A video sequence can be considered as a sequence of images sampled from an *appearance manifold*, which contains all possible appearances of the subject [17]. The complex nonlinear appearance manifold of each subject can be viewed by a collection of submanifolds, where each submanifold models the face appearances of the person in nearby poses. Each submanifold is in turn approximated by a low-dimensional linear subspace, which can be computed by PCA on training video sequences. Given a video sequence, the temporal appearance variation is modeled as transitions between these subspaces and these can be learned directly from the training data. Further, the tracking problem can also be integrated into this framework by searching for a bounding-box on the test image that minimizes the distance of the cropped region to the learnt appearance manifold. The recognition module uses the subimage returned by the tracker to compute its distance from the appearance manifold for each person and a maximum likelihood estimate yields the recognition result for each frame. This method is shown to be quite robust to large appearance changes if sufficient illumination and pose variations are available in the training video sequences.

In a related work, Arandjelovic and Cipolla [18] propose a *Shape-Illumination manifold* to describe the appearance variations due to the combined effects of facial shape and illumination. They consider a weak photometric model of image formation in which the pixel intensity is a linear function of the albedo $\boldsymbol{\rho}(i)$ of the corresponding 3D point

$$\boldsymbol{I}(i) = \boldsymbol{\rho}(i) \cdot \boldsymbol{s}(i) \tag{8}$$

where $\boldsymbol{s}$ is a function of illumination, shape, and other parameters which are not explicitly modeled. If $\boldsymbol{I}_1$ and $\boldsymbol{I}_2$ be two images of the same person, in the same pose, but different illuminations,

$$\Delta \log \boldsymbol{I}(i) = \log \boldsymbol{s}_2(i) - \log \boldsymbol{s}_1(i) \tag{9}$$

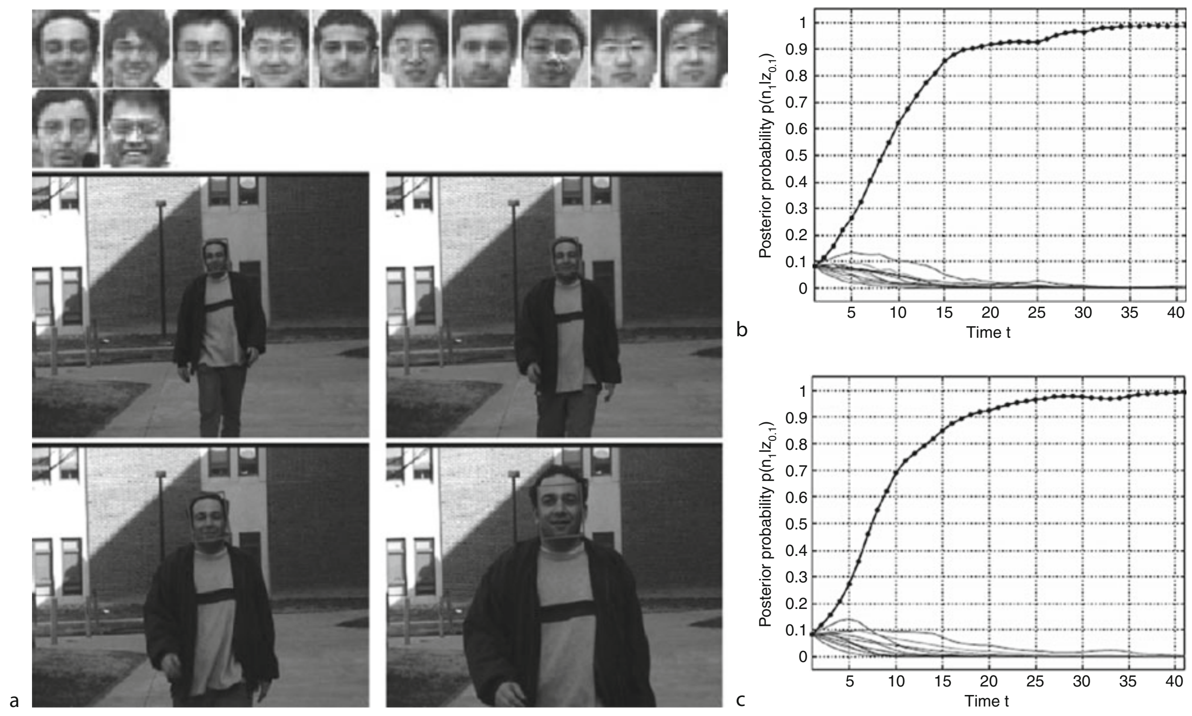
The difference between these logarithm-transformed images does not depend on the facial albedo and as the pose of the subject varies, it describes a manifold which is termed here as the Shape-Illumination manifold. They assume that the Shape-Illumination Manifold of all possible illuminations and head poses is generic for human faces and can be learnt effectively using Probabilistic PCA from a small, unlabeled set of video sequences of faces taken in randomly varying lighting conditions. Given a novel sequence, the learnt model is used to decompose the face appearance manifold into albedo and shape-illumination manifolds, producing the classification decision by robust likelihood estimation.

Face recognition from video sequence involves both tracking the faces and then recognizing the tracked faces. In the conventional tracking-then-recognition approach, where tracking and recognition is performed sequentially and separately, the criteria for selection of good frames, parameter estimation, etc., is often done in a heuristic manner. By effectively exploiting temporal information, the tracking-and-recognition framework performs both the tasks simultaneously in a unified probabilistic framework [19]. To fuse temporal information, the time series state space model is used to characterize the evolving kinematics and identity in the probe video. Depending on whether there is significant pose variation in the video sequence or not, 3D motion parameters or affine motion parameters can be used accordingly. The identity equation governs the temporal evolution of the identity variable and assumes that the identity does not change as time proceeds. The observation is assumed to be a transformed and noise-corrupted version of some still template in the gallery. This transformation could be either geometric or photometric or both. Using the sequential importance sampling (SIS) technique, the joint posterior distribution of the motion vector and the identity variable is estimated at each time instant and then propagated to the next time instant using the motion and identity equations. The marginal distribution of the identity variable is estimated to provide the recognition result. The still templates in the gallery can be generalized to video sequences for video-to-video recognition. Top row in Fig. 3a shows a few gallery images and the bottom row shows a few frames from tracking results in a still-to-video recognition scenario where the gallery consists of still images and the probe is a video sequence. Figure 3b shows how the posterior probability of identity of a person changes over time. The posterior probability for the correct identity increases as time proceeds and eventually approaches 1, and all others go to 0.

The dynamic information present in the video sequence can also be learnt and utilized for face recognition using a hidden Markov model (HMM) [20, 21]. A HMM is a statistical model used to characterize the statistical properties of a signal and has been successfully applied to model temporal information for speech recognition, gesture recognition, expression recognition, etc. Given the video sequences of the subjects in the gallery, the statistics of training video sequences and the temporal dynamics of each subject are learned by an HMM. For training, each frame in the video sequence is projected to low-dimensional space using PCA and then used as the observation. During recognition, the temporal characteristics of the test video sequence are analyzed over time by the HMM corresponding to each subject. As done for the database sequences, all frames of the test video sequence are projected into the eigenspace and the resulting feature vectors are used as observation vectors. Given the computed HMM of the database subjects, the likelihood score of the observation vectors is computed and the highest score provides the identity of the test video sequence. Furthermore, with unsupervised learning, each HMM can be adapted with the test video sequence, which results in better modeling over time.

Open Problems

The wide range of applications has made face recognition both from still images and video sequences very important areas of research. Though significant efforts have gone into understanding the different sources of variations affecting the facial appearance, the accuracy of face recognition algorithms in completely uncontrolled scenarios is still far from satisfactory. In the Face Recognition Grand Challenge 2006 [22], it was observed that many algorithms perform almost flawlessly (verification rate of over 0.99 for FAR of 0.001) for images taken under controlled illumination, but the performance significantly deteriorates with uncontrolled query images (verification rate of around 0.8 for FAR of 0.001). Pose and illumination variations still remain one of the biggest challenges for face recognition. Most of the approaches that can handle noncanonical facial pose require manual labeling of several landmark locations in the face, which is an impediment to realizing a completely automatic face recognition system. Another issue that needs to be addressed is robustness to low resolution images. Standard face recognition methods work well if the images or videos are of high quality, but usually fail when enough resolution is not available. Other than these variations, a lot of work is needed to handle the other sources of variations like expressions, make-up, aging, etc. Face recognition from video sequences is a relatively new research area as compared to still image-based face recognition and there are many unaddressed issues. The large amount of data that is being collected from surveillance cameras requires efficient methods for acquiring, preprocessing and analyzing these video streams. Also, much work needs to be done to optimally use the dynamic information present in the video sequence for recognition. Another area of current research is in generating secure face signatures, known as cancelable biometrics [23]. Finally, ideas from studies on



Face Recognition from Still Images and Video. Fig. 3 (**a**) Top: 2D appearance models for the subjects in the gallery. Bottom: A few images from a video sequence of a person walking. The face is being tracked and the tracked face is matched with the 2D appearance models in the gallery for recognition. (**b**) Posterior probability of identity against time t, obtained by the CONDENSATION algorithm and (**c**) the proposed algorithm (courtesy Zhou et al. [19])

human perception of faces [24] must be integrated in the design of algorithms for face recognition.

Recommended Reading

1. Kanade T (1973) Computer Recognition of Human Faces. Birkhauser, Basel and Stuttgart
2. Kelly MD (1970) Visual identification of people by computer. Technical Report AI-I30, Stanford AI Project, Standard, CA
3. Turk M, Pentland A (1991) Eigenfaces for recognition. J Cogn Neurosci 3(1):71–86
4. Belhumeur PN, Hespanha JP, Kriegman DJ (1997) Eigenfaces vs. fisherfaces: recognition using class specific linear projection. IEEE transactions on pattern analysis and machine intelligence 19(7):711–720
5. Etemad K, Chellappa R (1997) Discriminant analysis for recognition of human face images. J Opt Soc Am A 14(8): 1724–1733
6. Wiskott L, Fellous JM, Kruger N, Malsburg C (1997) Face recognition by elastic bunch graphmatching. IEEE transactions on pattern analysis and machine intelligence 19(7): 775–779
7. Jacobs DW, Belhumeur PN, Basri R (1998) Comparing images under variable illumination. IEEE conference on computer vision and pattern recognition 610–617
8. Zhou SK, Aggarwal G, Chellappa R, Jacobs DW (2007) Appearance characterization of linear lambertian objects, generalized photometric stereo, and illumination-invariant face recognition. IEEE transactions on pattern analysis and machine intelligence 29(2):230–245
9. Biswas S, Aggarwal G, Chellappa R (2009) Robust estimation of albedo for illumination-invariant matching and shape recovery. IEEE transactions on pattern analysis and machine intelligence 31(5):884–899
10. Blanz V, Vetter T (2003) Face recognition based on fitting a 3D morphable model. IEEE transactions on pattern analysis and machine intelligence 25(9):1063–1074
11. Basri R, Jacobs DW (2003) Lambertian reflectance and linear subspaces. IEEE transactions on pattern analysis and machine intelligence 25(2):218–233
12. Ramamoorthi R, Hanrahan P (2001) On the relationship between radiance and irradiance: determining the illumination from images of convex Lambertian object. J Opt Soc Am A 2448–2459
13. Zhang L, Samaras D (2006) Face recognition from a single training image under arbitrary unknown lighting using spherical harmonics. IEEE transactions on pattern analysis andmachine intelligence 28(3):351–363
14. Yue Z, Zhao W, Chellappa R (2006) Pose-encoded spherical harmonics for face recognition and synthesis using a single image. EURASIP J Adv Signal Process 1–18
15. Kumar N, Berg AC, Belhumeur PN, Nayar SK (2009) Attribute and simile classifiers for face verification. IEEE international conference on computer vision

16. O'Toole AJ, Roark D, Abdi H (2002) Recognizing moving faces: a psychological and neural synthesis. Trends Cogn Sci 6:261–266
17. Lee K, Ho J, Yang M, Kriegman DJ (2005) Visual tracking and recognition using probabilistic appearance manifolds. Comp Vis Image Und 99(3):303–331
18. Arandjelovic O, Cipolla R (2006) Face recognition from video using the generic shape-illuminationmanifold. Proceedings of European conference on computer vision LNCS 3954:27–40
19. Zhou S, Krueger V, Chellappa R (2003) Probabilistic recognition of human faces from video. Comp Vis Image Und (special issue on Face Recognition) 91:214–245
20. Liu X, Cheng T (2003) Video-based face recognition using adaptive hidden Markov models. IEEE proceedings of computer vision and pattern recognition 1:340–345
21. Phillips PJ, Flynn PJ, Scruggs T, Bowyer KW, Worek W (2006) Preliminary face recognition grand challenge results. Proc. seventh int'l conf. automatic face and gesture recognition 15–24
22. Phillips PJ, Flynn PJ, Scruggs T, Bowyer KW, Worek W (2006) Preliminary face recognition grand challenge results. Proceedings of automatic face and gesture recognition 15–24
23. Ratha NK, Chikkerur S, Connell JH, Bolle RM (2007) Generating cancelable fingerprint templates. IEEE transactions on pattern analysis and machine intelligence 29(4):561–572
24. Sinha P, Balas BJ, Ostrovsky Y, Russell R (2006) Face recognition by humans: 19 results all computer vision researchers should know about. Proceedings of the IEEE 94(11):1948–1962

Factor Base

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

Related Concepts

►Discrete Logarithm Problem; ►Function Field Sieve; ►Index Calculus Method; ►Number Field Sieve for the DLP; ►Number Field Sieve for Factoring; ►Quadratic Sieve; ►Smoothness

Definition

A *factor base* is a set of values among which relations are constructed in certain algorithms for solving number theory–based problems.

Applications

The term *factor base* refers to the set of small prime numbers (and more generally, prime powers and irreducible polynomials) among which relations are constructed in various integer factoring algorithms (►Sieving) as well as in certain algorithms for solving the discrete logarithm problem (►Index Calculus).

The choice of factor base – and specifically the upper bound on the values in the factor base, called the *smoothness bound* – plays a crucial role in the time and hardware requirements for these algorithms. For further discussion, see ►smoothness.

Factorization Circuits

Daniel J. Bernstein
Department of Computer Science, University of Illinois at Chicago, Chicago, Illinois, USA

Related Concepts

►Integer Factorization

Definition

Factorization circuits are circuits specially designed to factor integers.

Theory and Applications

There is a long history of special-purpose machines built for ►integer factorization, including the machine à congruences (Carissan, 1919), the bicycle chain sieve (Lehmer, 1926), the Georgia Cracker (Smith and Wagstaff, 1982), CAIRN 1 [10], CAIRN 2 [3] and CAIRN 3 [4]. There is also a long history of designs that have been published at various levels of detail but that have not been reported to have been built, including Quasimodo [8], TWINKLE [5], a mesh-sieving circuit [1], an ECM circuit [1], ►TWIRL [9], and SHARK [2].

Application-specific integrated electronic circuits are generally believed to be the most cost-effective current technology for large-scale well-funded cryptanalytic computations. Almost all recent proposals are factorization-specific very-large-scale integrated electronic circuits. TWINKLE is an exception, using optical components together with electronic circuits, but does not appear to be more cost-effective than electronic circuits alone. CAIRN uses FPGAs to simulate factorization circuits.

The basic advantage of factorization circuits, compared to general-purpose computers running factorization software, is that circuits perform many more useful operations in parallel. However, each operation has fast access to only a very small amount of nearby storage; accessing large amounts of storage is relatively expensive. Optimizing the price-performance ratio (area-time product) of a circuit for integer factorization is not the same problem as optimizing the number of bit operations. See generally [1, 11].

The Georgia Cracker and Quasimodo use the ►quadratic sieve (QS). TWINKLE, the ECM circuit, TWIRL, SHARK, and CAIRN use the ►number field sieve (NFS). In both QS and NFS, the primary bottleneck is a relation-finding stage: considering a very large collection of small auxiliary integers and identifying the ►smooth integers. A secondary bottleneck is a matrix stage: finding linear relations in a matrix modulo 2.

Factorization circuits vary primarily in how they take advantage of parallelism for smoothness detection. The asymptotic scalability of these techniques is indicated by the time they take on a circuit of area $y^{1+o(1)}$ to recognize y-smooth integers in a collection of $y^{2+o(1)}$ auxiliary integers. The Georgia Cracker, Quasimodo, and TWINKLE use parallel versions of trial division and take time $y^{2+o(1)}$. General-purpose computers performing ►sieving in RAM also take time $y^{2+o(1)}$. The mesh-sieving circuit and TWIRL instead use parallel versions of sieving and take time $y^{1.5+o(1)}$. The ECM circuit and SHARK instead use the elliptic curve method and take time only $y^{1+o(1)}$.

Factorization circuits vary secondarily in how they take advantage of parallelism for the matrix step. "RAM" techniques take time $y^{2+o(1)}$ on a circuit of area $y^{1+o(1)}$ to find linear dependencies in a sparse matrix of size $y^{1+o(1)}$, while "mesh" techniques take time $y^{1.5+o(1)}$ for the same task.

The price–performance ratio for NFS is $\exp((\beta + o(1))(\log n)^{1/3}(\log\log n)^{2/3})$. Here n is the integer being factored, and β is shown in the following table:

Smoothness Detection	Matrix	Parameters	β
RAM sieving/TWINKLE	RAM	ops-optimized	2.85 . . .
RAM sieving/TWINKLE	RAM	β-optimized	2.76 . . . (2002 Pomerance)
mesh sieving/TWIRL	RAM	β-optimized	2.37 . . . (2001 Bernstein)
mesh sieving/TWIRL	mesh	β-optimized	2.36 . . .
ECM	RAM	β-optimized	2.08 . . .
ECM	mesh	β-optimized	1.97 . . . (2001 Bernstein)

Note that the overall cost of NFS is influenced much more by smoothness-detection parallelism than by matrix parallelism. Optimizing β often requires reducing the NFS smoothness bound, increasing the overall cost of smoothness detection but decreasing the overall cost of the matrix step.

More detailed analyses, such as [6, 7], concluded that 1024-bit integers could be factored in a year by various circuits with estimated costs between $\$10^6$ and $\$10^9$. RSA Laboratories and the US National Institute of Standards and Technology, citing these analyses, recommended switching from 1024-bit ►RSA to 2048-bit RSA by the end of 2010. As of January 2011, some Web sites such as https://www.google.com are continuing to use 1024-bit RSA, so the exact cost of 1024-bit factorization circuits remains of interest in applied cryptography.

Recommended Reading

1. Bernstein DJ (2001) Circuits for integer factorization: a proposal. URL: http://cr.yp.to/papers.html#nfscircuit
2. Franke J, Kleinjung T, Paar C, Pelzl J, Priplata C, Stahlke C (2005) Shark: a realizable special hardware sieving device for factoring 1024-bit integers. In: Rao JR, Sunar B (eds) Cryptographic hardware and embedded systems – CHES 2005, 7th international workshop, Edinburgh, 29 August–1 September 2005. Lecture notes in computer science, vol 3659. Springer, pp 119–130
3. Izu T, Kogure J, Shimoyama T (2007) CAIRN 2: an FPGA implementation of the sieving step in the number field sieve method. In: Paillier P, Verbauwhede I (eds) Cryptographic hardware and embedded systems – CHES 2007, 9th International workshop, Vienna, 10–13 September 2007. Lecture notes in computer science, vol 4727. Springer, pp 364–377
4. Izu T, Kogure J, Shimoyama T (2010) CAIRN: Dedicated integer factoring devices. In: International conference on network-based information systems, Takayama, pp 558–563
5. Lenstra AK, Shamir A (2000) Analysis and optimization of the TWINKLE factoring device. In: Preneel B (ed) Advances in cryptology – EUROCRYPT 2000, international conference on the theory and application of cryptographic techniques, Bruges, 14–18 May 2000. Lecture notes in computer science, vol 1807. Springer, pp 35–52
6. Lenstra AK, Shamir A, Tomlinson J, Tromer E (2002) Analysis of Bernstein's factorization circuit. In: Zheng Y (ed) Advances in cryptology – ASIACRYPT 2002, 8th international conference on the theory and application of cryptology and information security, Queenstown, 1–5 December 2002. Lecture notes in computer science, vol 2501. Springer, pp 1–26
7. Lenstra AK, Tromer E, Shamir A, Kortsmit W, Dodson B, Hughes J, Leyland PC (2002) Factoring estimates for a 1024-bit RSA modulus. In: Laih C-S (ed) Advances in cryptology – ASIACRYPT 2003, 9th international conference on the theory and application of cryptology and information security, Taipei, 30 November–4 December 2003. Lecture notes in computer science, vol 2894. Springer, pp 55–74
8. Pomerance C, Smith JW, Tuler R (1988) A pipeline architecture for factoring large integers with the quadratic sieve algorithm. SIAM J Comput 17(2):387–403
9. Shamir A, Tromer E (2003) Factoring large numbers with the TWIRL device. In: Boneh D (ed) Advances in cryptology – CRYPTO 2003, 23rd Annual international cryptology conference, Santa Barbara, 17–21 August 2003, Proceedings. Lecture notes in computer science, vol 2729. Springer, pp 1–26
10. Shimoyama T, Izu T, Kogure J (2005) Implementing a sieving algorithm on a dynamic reconfigurable processor (extended abstract). In: Special-purpose hardware for attacking cryptographic systems – SHARCS'05, Paris. http://www.hyperelliptic.org/tanja/SHARCS/talks/SHARCS2005paper-shimoyama.pdf
11. Wiener MJ (2004) The full cost of cryptanalytic attacks. J Cryptol 17:105–124

Fail-Stop Signature

Gerrit Bleumer
Research and Development, Francotyp Group,
Birkenwerder bei Berlin, Germany

Related Concepts

►Digital Signature; ►Forgery

Definition

Fail-stop signatures are ►digital signatures where signers enjoy unconditional unforgeability (with an unavoidable but negligible error probability), while the verifiers bear the risk of forged signatures and therefore enjoy computational ►security only. If a signer is confronted with an alleged signature that she has not produced, then the signer can with overwhelming probability prove that the alleged signature is in fact forged. Afterward, the signer can revoke her verifying key, thus the name fail-stop signature scheme.

Background

Fail-stop signatures were introduced by Pfitzmann [4] who gives an in-depth introduction in [6]. The security for the signer is strictly stronger than the strongest security defined by Goldwasser, Micali, and Rivest (►GMR signatures [2]), where signers enjoy computational security, while verifiers are unconditionally secure against ►forgery. In other words, the signer is secure even against counterfeiting by a computationally unrestricted attacker. In the same sense, fail-stop signatures provide strictly stronger security for the signer than all the standard digital signature schemes such as ►RSA digital signatures, ►Rabin digital signatures, ►ElGamal digital signatures, ►Schnorr digital signatures, Digital Signature Algorithm (DSA), ►elliptic curve signature schemes (ECDSA), etc. A comparative overview is found in [5]. Fail-stop signatures are more fundamental than ordinary digital signatures because the latter can be constructed from the former, but not vice versa [3].

Theory

In a fail-stop signature scheme, each signing member has an individual signing key pair of a private signing key and a public verifying key (►public key cryptography). The verifying algorithm takes the signer's public verifying key, a message, and a corresponding signature and returns a Boolean result indicating whether the signature is valid for the message with respect to the public verifying key. If a signer is confronted with a signature that she did not produce, but which holds against the verifying algorithm, then the signer can use an additional algorithm to produce a proof of forgery. Any third party verifier can then verify by means of an additional algorithm that the proof of forgery is valid. The first efficient fail-stop signature scheme was constructed by van Heijst, Pedersen, and Pfitzmann [3, 10, 11].

Pedersen and Pfitzmann have shown that fail-stop signatures can be produced and verified about as efficiently as ordinary ►digital signatures, but that the length of the signer's private signing key is proportional to the number of messages that can be signed with it [3]. However, signers need not store complete long signing keys at any one time, but can generate random bits for them on demand.

Fail-stop signatures have been proposed to be used in ►electronic cash schemes such that customers need not bear the risk of very powerful banks forging some of their customer's signatures. Although this may appear far fetched for the case of general customers, it may be a necessary condition to open the door for processing high-value transactions (millions of dollars) fully electronically.

A fail-stop signature scheme has the following operations: (1) an operation for generating pairs of a private signing key and a public verifying key for an individual, (2) an operation to produce fail-stop signatures, (3) an operation to verify fail-stop signatures, (4) an operation to produce proofs of forgery, and (5) an operation to verify proofs of forgery.

The characteristic security requirements of a fail-stop signature scheme are:

- *Unforgeability* (*Recipient's Security*): resistance against ►existential forgery under adaptive chosen message attacks by computationally restricted attackers.
- ►*Nonrepudiation*: a computationally restricted signer cannot produce a fail-stop signature that he can later prove to be a forged signature.
- *Signer's Security*: if a computationally unlimited attacker fabricates a signature, the alleged signer can produce a valid proof of forgery with overwhelming probability.

Constructions have been based on ►groups, in which the ►discrete logarithm problem is hard [3, 10, 11], on the ►RSA digital signature scheme [7], and on ►authentication codes [9].

Fail-stop signatures schemes can be equipped with additional features: Chaum et al. have proposed undeniable fail-stop signatures [1]. Susilo et al. have proposed threshold fail-stop signatures [8].

Applications

In some applications of unconditionally secure message authentication codes, fail-stop signatures can be used as a replacement in order to increase the computational efficiency (e.g., ►DC-Networks).

Recommended Reading

1. Chaum D, van Heijst E, Pfitzmann B (1992) Cryptographically strong undeniable signatures, unconditionally secure for the signer. In: Feigenbaum J (ed) Advances in cryptology – CRYPTO'91. Lecture notes in computer science, vol 576. Springer, Berlin, pp 470–484
2. Goldwasser S, Micali S, Rivest RL (1988) A digital signature scheme secure against adaptive chosen-message attacks. SIAM J Comput 17(2):281–308
3. Pedersen TP, Pfitzmann B (1997) Fail-stop signatures. SIAM J Comput 26(2):291–330
4. Pfitzmann B (1991) Fail-stop signatures; principles and applications. In: Proceedings COMPSEC'91, 8th world conference on computer security, audit and control. Elsevier, Oxford, pp 125–134
5. Pfitzmann B (1993) Sorting out signature schemes. 1st ACM conference on computer and communications security, Fairfax, November 1993. ACM Press, New York, pp 74–85
6. Pfitzmann B (1996) Digital signature schemes general framework and fail-stop signatures. In: Pfitzmann B (ed) Lecture notes in computer science, vol 1100. Springer, Berlin
7. Susilo W, Safavi-Naini R, Pieprzyk J (1999) RSA-based fail-stop signature schemes. International workshop on security (IWSEC'99). IEEE Computer Society, Los Alamitos, CA, pp 161–166
8. Susilo W, Safavi-Naini R, Pieprzyk J (1999) Fail-stop threshold signature schemes based on elliptic curves. In: Pieprzyk J, Safari-Naini R, Seberry J (eds) Information security and privacy. Lecture notes in computer science, vol 1587. Springer, Berlin, pp 103–116
9. Susilo W, Safavi-Naini R, Gysin M, Seberry J (2000) A new and efficient fail-stop signature scheme. Comput J 43(5):430–437
10. van Heijst E, Pedersen TP (1993) How to make efficient fail-stop signatures. In: Rueppel RA (ed) Advances in cryptology – EUROCRYPT'92. Lecture notes in computer science, vol 658. Springer, Berlin, pp 366–377
11. van Heijst E, Pedersen TP, Pfitzmann B (1993) New constructions of fail-stop signatures and lower bounds. In: Brickell EF (ed) Advances in cryptology – CRYPTO'92. Lecture notes in computer science, vol 740. Springer, Berlin, pp 15–30

Fair Exchange

Matthias Schunter
IBM Research-Zurich, Rüschlikon, Switzerland

Synonyms

Barter; Escrow service; Simultaneous transactions

Related Concepts

►Certified Mail; ►Contract Signing; ►Non-Repudiation

Definition

An exchange ►protocol transmits a digital ►token from each participant to each other participant in a group. An exchange protocol is called fair, if and only if

1. The protocol either ensures that all honest participants receive all items as expected or releases no useful information about any item of any honest party.
2. The protocol terminates after a fixed time.

Background

Fair exchange was introduced as fair exchange of secrets [3] and later generalized to fair exchange of a wider range of digital items [1].

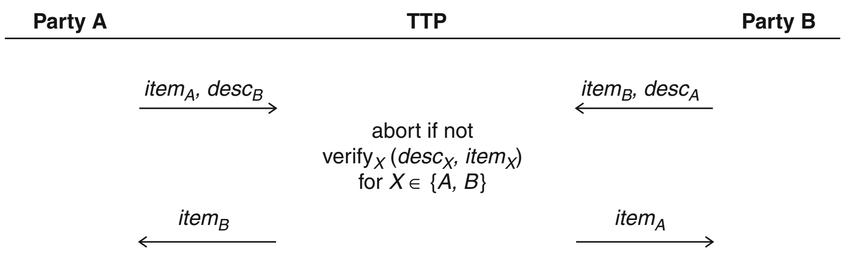
Theory

In the two-party case, fairness means that an honest party releases its item if it receives the expected item from the peer. Otherwise, no additional information is released. A practical example is Ebay's escrow service that works as follows: the buyer deposits a payment at Ebay, the seller sends the item to the buyer, and if the buyer approves, Ebay forwards the payment to the seller. Note that the exact formalization of "fixed time" depends on the network model: in synchronous networks with a global notion of rounds this is formalized by "fixed number of rounds." In asynchronous networks without global time, a maximum logical time [7] is used.

Early research focused on two-party protocols (multiparty ►protocols) solving particular instances of fair exchange such as ►contract signing [2] and [4] or ►certified mail [8]. However, these protocols are either unfair with a probability that is linear in the number of rounds (e.g., 1/10 for 10 rounds) or cannot guarantee termination in a fixed time [5].

Fairness with a negligible error probability within a limited time can only be guaranteed by involving a ►Trusted Third Party (TTP) or by assuming a trusted majority. A TTP ensures that the outcome is fair even if one party cheats. The TTP solves the fundamental problem that a party sending the complete item while not having received a complete item is always at disadvantage. Third parties can either be on-line, i.e., involved in each protocol run or optimistic, i.e., involved only if an exception occurs.

Generic fair exchange protocols [1] can exchange many types of items. Examples are data (described by a publicly known property), receipts, signatures on well-known data,



Fair Exchange. Fig. 1 A generic fair exchange protocol with on-line trusted third party

or payments. Besides generic protocols, there exist a variety of protocols that optimize or extend exchanges of particular items: payment for receipt, payment for data ("fair purchase"), data for receipt ("certified mail"), or signature for signature ("contract signing").

A nonoptimistic generic fair exchange protocol similar to the contract signing protocol in [9] is depicted in Fig. 1: each party, say A, sends its item $item_A$ and a description $desc_B$ of the item expected in return to the TTP. The TTP exchanges the items if the expectations are met. This is determined by a function verify() that verifies that an item matches its description. Otherwise, the exchange is aborted and no additional information about the items is released. Fair exchange protocols can be further classified by the number of participants, the network model they assume, the properties of the items they produce, and whether they are abuse-free or not. In synchronous networks, there exists a global notion of rounds that limits the time needed to transmit messages between correct parties. In asynchronous networks, messages between honest participants are guaranteed to be delivered eventually; but the transmission time is unbounded. Besides these two well-known network models, some protocols assume special network models that assume trusted hosts such as reliable messaging boards or reliable file-transfer servers. Abuse-free protocols [6] provide the additional guarantee that if a failed protocol runs it never produces any evidence. This is of some importance for contract signing where a dishonest party may gain an advantage by being able to prove that a particular honest party was willing to sign a particular contract.

Recommended Reading

1. Asokan N, Schunter M, Waidner M (1997) Optimistic protocols for fair exchange. 4th ACM conference on computer and communications security, Zürich, April 1997. ACM Press, New York, pp 6–17
2. Blum M (1981) Three applications of the oblivious transfer, version 2. Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley
3. Blum M, Vazirani UV, Vazirani VV (1984) Reducibility among protocols (extended abstract). Advances in cryptology – CRYPTO'83. Plenum, New York, pp 137–146
4. Even S (1983) A protocol for signing contracts. ACM SIGACT News 15(1):34–39
5. Even S, Yacobi Y (1980) Relations among public key signature systems. Technical report Nr. 175, Computer Science Department, Technion, Haifa, Israel
6. Garay JA, Jakobsson M, MacKenzie P (1999) Abuse-free optimistic contract signing. In: Wiener J (ed) Advances in cryptology – CRYPTO'99. Lecture notes in computer science, vol 1666. Springer, Berlin, pp 449–466
7. Lynch NA (1966) Distributed algorithms. Morgan Kaufmann, San Francisco
8. Rabin MO (1981) Transaction protection by beacons. Aiken Computation Laboratory, Harvard University, Cambridge, MA. Technical report TR-29-81
9. Rabin M (1983) Transaction protection by beacons. J Comput Syst Sci 27:256–267

False Data Filtering

►False Data Injection Defense

False Data Injection Defense

Wensheng Zhang
Department of Computer Science, College of Liberal Arts and Sciences Iowa State University, Ames, IA, USA

Synonyms

False data filtering

Definition

False data injection refers to techniques that can detect and filter out false data injected by the adversary to wireless networks so as to save the bandwidth and energy that would be consumed to transmit such data.

Background

In a sensor network, sensor nodes need to report sensory data to a sink through multihop forwarding. Due to the open nature of wireless communication, the adversary can easily inject false data or modify data forwarded in the network. Public key cryptographic signatures could be used to authenticate each data message so as data forwarders and receivers are able to detect false or modified data. However, performing public key cryptographic operations at every data forwarder, which is typically a resource-constrained node, may significantly increase data forwarding delay and energy consumption. Hence, more efficient solutions are needed to defend against the attacks.

Theory and Application

A number of symmetric cryptography-based schemes have been proposed to address the false data injection problem. Ye et al. [6] proposed a statistical en-route filtering (SEF) scheme. With this scheme, the network is associated with a global key pool, which consists of n partitions. Each partition 1contains m different keys. Each sensor node randomly picks k secret keys from one of the n partitions. When a node detects some event and wants to report, it generates a message authentication code (MAC) using one of its k keys to authenticate its report. The report is further endorsed by $T-1$ nodes in the same cluster of the detecting node, and each of the T nodes has keys from a distinct partition. If a false data message is injected to the network, or an endorsed data message is altered when being forwarded in the network, each forwarding node has the probability of $\frac{k}{n \cdot m}$ to detect it. The SEF scheme is independent of the topology changes in the sensor network. However, T is a security threshold. In the worst case, the SEF scheme can tolerate only $T-1$ compromised nodes in the network. Follow-up research has been conducted to improve the resilience to node compromise. Among such efforts, Zhang and Cao [9] proposed localized key updating schemes to periodically update the keys that are used in endorsing reports, and Yang et al. [5] proposed to bind keys with locations to mitigate the effect of node compromise.

Zhu et al. [10] proposed an interleaved hop-by-hop scheme. In this scheme, each sensor node is in a cluster of at least $t+1$ members and the cluster has a path toward the sink. Along the path, two nodes that are $t+1$ hops away are associated by establishing a unique pairwise key. When a sensor node detects some event and wants to report, the detecting node itself and t other nodes of its cluster endorse the report by generating and attaching $t+1$ MACs to the report. The MAC generated by each node uses the pairwise key shares between the node and its associated node that is $t+1$ hops away toward the sink. When the endorsed message is forwarded, each forwarding node verifies the MAC generated by its upstream associated node. If verification succeeds, it replaces the MAC with a new one using its pairwise key shared with its downstream associated node. Since the pairwise key shared by each pair of associated nodes is distinct, this scheme can tolerate up to t compromised nodes in each cluster or consecutive nodes on a path, instead of over the whole network. However, if the network topology changes, the association between nodes need to be updated accordingly.

To defend against false data injection in broadcast, authentication scheme μTESLA [1] can be applied under the condition that sensor nodes are time synchronization and nodes share a commitment with the broadcasting node. Schemes based on public key and probabilistic verification [3] have also been proposed to filter false data injection in broadcast.

Identifying and removing "moles" which inject false data is more desirable than passively filtering false data injection. Ye et al. [7] proposed probabilistic nested marking (PNM) scheme to locate moles. The design is based on the following basic nested marking idea: Each forwarding node V_i appends to the packet its ID i and a MAC computed with its secret key k_i known only to the sink. This way, each forwarding node indicates its presence on the route and shows to the sink what it has received. As the appended information is nested, any tampering with the packet by a malicious forwarding node will make the MAC invalid to the sink, and where the tampering takes place can also be traced out. The PNM scheme extends the above idea by allowing each forwarding node to mark the packet with only a small probability, while the moles can still be located by the sink. The PNM scheme, however, does not allow forwarding nodes to identify and filter false data injection. This can be addressed with the packet dropper and modifier catching scheme proposed by Wang et al. [2], where false data injection can be filtered by forwarding nodes, and malicious nodes who modify packets can be identified as nodes who drop packets.

Open Problems

More efficient and scalable approaches to filter data injection in broadcast is demanded. The existing schemes [1, 3] either requires time synchronization or relies on public key cryptographic operations, which lack scalability or efficiency.

Recommended Reading

1. Perrig A, Szewczyk R, Tygar J, Wen V, Culler D (2001) Spins: security protocols for sensor networks. Proceedings of the annual international conference on mobile computing and networking (MobiCom), pp 189–199
2. Wang C, Feng T, Kim J, Wang G, Zhang W (2009) Catching packet droppers and modifiers in wireless sensor networks. Proceedings of IEEE SECON
3. Wang R, Du W, Ning P (2007) Containing denial-of-service attacks in broadcast authentication in sensor networks. Proceedings of ACM symposium on mobile ad hoc networking and computing (MobiHoc), pp 71–79
4. Yang H, Lu S (2004) Commutative cipher based en-route filtering in wireless sensor networks. Proceedings of IEEE vehicular technology conference, pp 1223–1227
5. Yang H, Ye F, Yuan Y, Lu S, Arbaugh W (2005) Towards resilient security in wireless sensor networks. Proceedings of ACM symposium on mobile ad hoc networking and computing (MobiHoc), pp 34–45
6. Ye F, Luo H, Lu S, Zhang L (2004) Statistical en-route filtering of injected false data in sensor networks. Proceedings of IEEE conference on computer communications (INFOCOM), pp 839–850
7. Ye F, Yang H, Liu Z (2007) Catching "moles" in sensor networks. Proceedings of the international conference on distributed computing systems (ICDCS), pp 66–69
8. Yu Z, Guan Y (2006) A dynamic en-route scheme for filtering false data injection in wireless sensor networks. Proceedings of IEEE conference on computer communications (INFOCOM)
9. Zhang W, Cao G (2005) Group rekeying for filtering false data in sensor networks: a predistribution and local collaboration based approach. Proceedings of IEEE conference on computer communications (INFOCOM), pp 503–514
10. Zhu S, Setia S, Jajodia S, Ning P (2004) An interleaved hop-by-hop authentication scheme for filtering false data in sensor networks. Proceedings of IEEE symposium on security and privacy, pp 1540–7993

Fast Correlation Attack

Anne Canteaut
Project-Team SECRET, INRIA Paris-Rocquencourt, Le Chesnay, France

Related Concepts

►Combination Generator; ►Correlation Attack; ►Filter Generator; ►Stream Cipher

Definition

Fast correlation attacks were first proposed by Meier and Staffelbach in 1988 [14, 15]. They apply to ►*running-key* generators based on ►*linear feedback shift registers (LFSRs)*, exactly in the same context as the ►*correlation attack*, but they are significantly faster. They rely on the same principle as the correlation attack: they exploit the existence of a correlation between the keystream and the output of a single LFSR, called the *target LFSR*, whose initial state depends on some bits of the secret key. In the original correlation attack, the initial state of the target LFSR is recovered by an exhaustive search. Fast correlation attacks avoid examining all possible initializations of the target LFSR by using some efficient error-correcting techniques. But, they require the knowledge of a longer segment of the keystream (in the context of a ►*known-plaintext attack*). As for the correlation attack, similar algorithms can be used for mounting a ►*ciphertext only attack* when there exists redundancy in the plaintext.

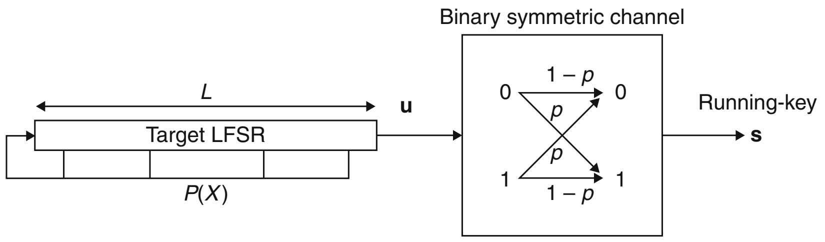
Theory

Fast correlation attacks as a decoding problem. The key idea of fast correlation attacks consists in viewing the correlation attack as a decoding problem. If there exists a correlation between the keystream $\mathbf{s}$ and the output $\mathbf{u}$ of the target LFSR, then the running-key subsequence $(s_t)_{t<N}$ can be seen as the result of the transmission of $(u_t)_{t<N}$ through the binary symmetric channel with error probability $p = \Pr[s_t \neq u_t] < 1/2$ (see Fig. 1). If $\Pr[s_t \neq u_t] > 1/2$, the bitwise complement of $\mathbf{s}$ is considered. Moreover, all bits of the LFSR sequence $\mathbf{u}$ depend linearly on the LFSR initial state, $u_0 \dots u_{L-1}$. Therefore, $(u_t)_{t<N}$ is a codeword of a linear code of length N and dimension L defined by the LFSR feedback polynomial. Thus, recovering the LFSR initial state consists in decoding the running-key subsequence relatively to the LFSR code.

With this formulation, the original ►*correlation attack* proposed by Siegenthaler consists in applying a maximum-likelihood decoding algorithm to the linear code defined by the LFSR feedback polynomial. It then requires testing the 2^L possible initial states of the target LFSR. The complexity can be reduced by using faster decoding algorithms. But, they usually require a larger number of running-key bits.

Decoding techniques for fast correlation attacks. Several algorithms can be used for decoding the LFSR code, based on the following ideas:

- Find many sparse linear recurrence relations satisfied by the LFSR sequence (these relations correspond to sparse multiples of the feedback polynomial), and use them in an iterative decoding procedure dedicated to low-density parity-check codes [3, 13, 15]. The complexity of this attack may significantly decrease when the feedback polynomial of the target LFSR is sparse. Thus, the use of sparse LFSR



Fast Correlation Attack. Fig. 1 Model for fast correlation attacks on LFSR-based stream ciphers

feedback polynomials should be avoided in LFSR-based running-key generators.

- Construct a convolutional code [6] (or a turbo code [7]) from the LFSR code, and use an appropriate decoding algorithm for this new code (Viterbi algorithm or turbo-decoding).
- Construct a new linear block code with a lower dimension from the LFSR code and apply to this smaller linear code a maximum-likelihood decoding algorithm [4, 11], or a polynomial reconstruction technique [8].

A survey on all these techniques and their computational complexities can be found in [10]. In practice, the most efficient fast correlation attacks enable to recover the initial state of a target LFSR of length 60 for an error-probability $p = 0.4$ in a few hours on a PC from the knowledge of 10^6 running-key bits.

Applications

Fast correlation attacks on combination generators. In the particular case of a combination generator, the target sequence $\mathbf{u}$ is the sequence obtained by adding the outputs of $(m+1)$ constituent LFSRs, where m is the correlation-immunity order of the combining function (►*correlation attack*). Thus, this sequence $\mathbf{u}$ corresponds to the output of a unique LFSR whose feedback polynomial is the greatest common divisor of the feedback polynomials of the $(m+1)$ involved LFSRs. Since the feedback polynomials are usually chosen to be primitive, the length of the target LFSR is the sum of the lengths of the $(m+1)$ LFSRs. The keystream corresponds to the received word as output of the binary symmetric channel with error-probability

$$p = \Pr[s_t \neq u_t] = \frac{1}{2} - \frac{1}{2^{n+1}}|\hat{\mathrm{f}}(t)|,$$

where n is the number of variables of the combining function, t is the n-bit vector whose ith component equals 1 if and only if $i \in \{i_1, i_2, \ldots, i_{m+1}\}$ and $\hat{\mathrm{f}}$ denotes the Walsh transform of f (►*correlation attack* and ►*Boolean functions*).

Fast correlation attacks on filter generators. In the case of a filter generator, the target LFSR has the same feedback polynomial as the constituent LFSR, but a different initial state. Actually, if the keystream is given by

$$s_t = f(v_{t+\gamma_1}, v_{t+\gamma_2}, \ldots, v_{t+\gamma_n}),$$

where $\mathbf{v}$ is the sequence generated by the constituent LFSR. The optimal target sequence $\mathbf{u}$ then corresponds to

$$u_t = \sum_{i=1}^{n} \alpha_i v_{t+\gamma_i}$$

where $\alpha = (\alpha_1, \ldots, \alpha_n)$ is the vector which maximizes the magnitude of the Walsh transform of the filtering function. Thus, the keystream corresponds to the received word as output of the binary symmetric channel with error-probability

$$p = \Pr[s_t \neq u_t] = \frac{\mathcal{NL}(f)}{2^n},$$

where $\mathcal{NL}(f)$ is the *nonlinearity* of the filtering function. The fast correlation attacks on filter generators can be improved by using several target LFSRs together [2, 9].

Other particular fast correlation attacks apply to filter generators, like conditional block-oriented correlation attacks [1, 5, 12] (►*filter generator*).

Recommended Reading

1. Anderson RJ (1995) Searching for the optimum correlation attack. In: Fast software encryption 1994. Lecture notes in computer science, vol 1008. Springer, pp 137–143
2. Canteaut A, Filiol E (May 2002) On the influence of the filtering function on the performance of fast correlation attacks on filter generators. In: Symposium on information theory in the Benelux, Louvain la Neuve, Belgium
3. Canteaut A, Trabbia M (2000)Improved fast correlation attacks using parity-check equations of weight 4 and 5. In: Advances in cryptology – EUROCRYPT 2000. Lecture notes in computer science, vol 1807. Springer, pp 573–588
4. Chepyshov V, Johansson T, Smeets B (2000) A simple algorithm for fast correlation attacks on stream ciphers. In: Fast software encryption 2000. Lecture notes in computer science, vol 1978. Springer, pp 181–195

5. Golić JDj (1996) On the security of nonlinear filter generators. In: Fast software encryption 1996. Lecture notes in computer science, vol 1039. Springer, pp 173–188
6. Johansson T, Jönsson F (1999) Improved fast correlation attack on stream ciphers via convolutional codes. In: Advances in cryptology – EUROCRYPT 1999. Lecture notes in computer science, vol 1592. Springer, pp 347–362
7. Johansson T, Jönsson F (1999) Fast correlation attacks based on turbo code techniques. In: Advances in cryptology – CRYPTO 1999. Lecture notes in computer science, vol 1666. Springer, pp 181–197
8. Johansson T, Jönsson F (2000) Fast correlation attacks through reconstruction of linear polynomials. In Advances in Cryptology – CRYPTO 2000. Lecture notes in computer science, vol 1880. Springer, pp 300–315
9. Jönsson F, Johansson T (2002) A fast correlation attack on LILI-128. Inf Process Lett 81(3):127–132
10. Jönsson F (2002) Some results on fast correlation attacks. PhD thesis, University of Lund, Sweden, 2002
11. Joux A (2009) Algorithmic cryptanalysis. Chapman & Hall/CRC, Boca Raton
12. Lee S, Chee S, Park S, Park S (1996) Conditional correlation attack on nonlinear filter generators. In: Advances in cryptology – ASIACRYPT 1996. Lecture notes in computer science, vol 1163. Springer, pp 360–367
13. Mihaljevic MJ, Fossorier MPC, Imai H (2000) A low-complexity and high performance algorithm for the fast correlation attack. In: Fast software encryption 2000. Lecture notes in computer science, vol 1978. Springer, pp 196–212
14. Meier W, Staffelbach O (1988) Fast correlation attacks on stream ciphers. In: Advances in cryptology – EUROCRYPT 1988. Lecture notes in computer science, vol 330. Springer, pp 301–314
15. Meier W, Staffelbach O (1989) Fast correlation attack on certain stream ciphers. J Cryptol 1:159–176

Fault Attack

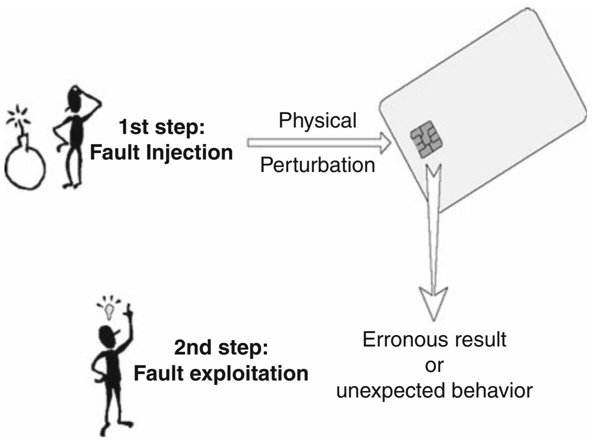
Olivier Benoît
Ingenico, Région de Saint-Étienne, France

Definition

A fault attack is an attack on a physicial electronic device (e.g., smartcard, HSM, USB token) which consists in stressing the device by an external mean (e.g., voltage, light) in order to generates errors in such a way that these errors leads to a security failure of the system (key recovery, ePurse balance increase, false signature acceptation, PIN code recovery…).

Introduction

A successful fault attack on an Integrated Circuit Card (ICC) or smartcard requires two steps: fault injection and the fault exploitation Fig. 1. The first step consists in injecting a fault at the appropriate time during the process. Fault injection is very dependent on the hardware and therefore the ICC. The second step consists in exploiting the erroneous result or unexpected behavior. Fault exploitation depends on the software design and implementation. In the case of an algorithm, it will also depend on its specification since the fault exploitation will be combined with cryptanalysis most of the time. Depending on the type of analysis performed, the fault injection will have to be done at a precise instant or roughly in a given period of time.



Fault Attack. Fig. 1 Fault attack process

Fault injection

There are many ways to generate a fault in an ICC. Already, three major means of fault injection can be distinguished:

Electrical perturbation on the standard ISO contact of the smartcard

- Vcc glitch (see below)
- Clock duty cycle and/or frequency alteration

Light-beam perturbation (contact-less)

- Global light-beam (wide spectrum)
- Focused light-beam (wide spectrum)
- Laser-beam (single wavelength)

Electromagnetic field perturbation (contact-less)

The effectiveness of each fault injection method strongly depends on the hardware design, manufacturing process, and technology. The chip behavior under a fault injection can be of four types:

- No effect
- Wrong results or unexpected behavior (exploitable fault)

- No response from the card (hardware reset required)
- Card is dead (physically damaged)

Of course, only one out of the four listed behaviors might be exploitable. Moreover, the perturbation can have a transient effect, permanent effect, or an intermediary state in between:

- Transient effect (only during fault injection)
- Semipermanent effect (for a variable period of time from a few minutes to a few days)
- Permanent effect

In practice, transient faults are easier to exploit as will be seen in the fault exploitation section.

The main difficulty of the fault injection step is to find the appropriate parameters of the perturbation for the ICC in consideration. Inappropriate parameters will not lead to an exploitable fault such as a wrong result or an unexpected behavior. Therefore, there is a risk to irremediably kill the chip. Besides, if the hardware implements some security sensors and/or protection mechanism, it will be even more difficult to inject an exploitable fault without triggering a security mechanism.

A Vcc glitch is defined by many parameters among which are its shape, the falling and rising slope, the low and high level, and the low-level duration. Very good results have been obtained on some IC with a limited control over these parameters. In fact, the three main parameters that needed to be tuned are the low- and high-voltage values and the glitch duration. Besides, the combination of all parameters quickly gives a huge search space and therefore it is not practical to have too many parameters to play with.

A white light perturbation can be very effective on some ICC. The hardware required for such an attack can be easy to set up by using a photo camera bulk and the associated electronic. Then, the major parameters to be tuned are the light emission intensity, the light emission duration and, eventually, the selection of a specific area of the chip to be exposed (focused light attack) rather than the complete die.

Using a laser such as a pulsed Nd:YAG laser is even more powerful because the energy level, the wavelength, the beam duration, and the beam size are much more under control. It is important to control the energy level in order to avoid destruction of the chip (too much energy) or an ineffective beam (not enough energy).

Fault Exploitation

Fault exploitation is the mandatory second step of a successful fault attack. The fault exploitation directly depends on the fault effect, the fault localization in time, and the target of the attack. Two major types of target can be distinguished for a fault attack:

- The operating system and application-sensitive process
- The cryptographic algorithm

A sensitive process is defined as a piece of code processing data that is known to the external world but should not be modified. The fault injection will precisely modify such data. By definition, if the data should not be modified, it means that modifying it will compromise the system security to some extent.

Differential fault analysis (DFA) mainly consists in analyzing an algorithm result (ciphertext) under regular condition and under abnormal condition for the same input (plaintext). The abnormal condition is usually obtained by fault injection during the process (transient fault) or before the process (permanent fault). Differential fault analysis has been widely studied from a theoretical point of view. In September 1996, three researchers from Bellcore identified a new attack against *plain* RSA (►RSA Digital Signature Scheme), [1], when performed with the ►Chinese Remainder Theorem. This attack was reported in a Bellcore press release entitled *New Threat Model Breaks Crypto Codes*, but no technical details were provided. Later, another researcher wrote a short memo that seemed to describe a more realistic attack. In the case of a computation error, the Bellcore researchers showed how to recover the secret factors p and q of the public RSA modulus n from *two* signatures of the same message: a correct one and a faulty one. Lenstra remarked that only the faulty signature was required. At the same time, Joye and Quisquater noted that Lenstra's observation could actually be applied to all RSA-type cryptosystems, including variants based on Lucas sequences (LUC) and ►elliptic curves (KMOV, Demytko) [2].

In conclusion, a fault attack is a threat for any secure token (whatever the form factor) and must be taken into consideration at all steps of the product design and specification. Countermeasures and protection can be designed both in hardware and in software to thwart such attacks.

Recommended Reading

1. Boneh D, DeMillo RA, Lipton RJ (1997) On the importance of checking cryptographic protocols for faults. In: Fumy W (ed) Advances in cryptology – eurocrypt'97. Lecture notes in computer science, vol 1233. Springer, Berlin, pp 37–51
2. Joye M, Lenstra AK, Quisquater J-J (1999) Chinese remaindering cryptosystems in the presence of faults. J Cryptol 12(4): 241–245

FEAL

Christophe De Cannière
Department of Electrical Engineering, Katholieke Universiteit Leuven, Leuven-Heverlee, Belgium

Related Concepts

►Block Ciphers; ►Differential Cryptanalysis; ►Feistel Cipher; ►Linear Cryptanalysis for Block Ciphers

The Fast Data Encipherment Algorithm (FEAL) [9] is a family of ►block ciphers developed by Shimizu and Miyaguchi. Since the introduction of its first version (FEAL-4, presented in 1987), the block cipher has stimulated the development of some of the most useful cryptanalytical techniques available today.

FEAL was designed to be a more efficient alternative to the ►Data Encryption Standard (DES), in particular in software. The block cipher encrypts data in blocks of 64 bits and uses a 64-bit key. It is a ►Feistel cipher, just as DES, but the components have been modified in order to be more suitable for word-oriented processors. The complete cipher can be implemented using only additions modulo 2^8, rotations over 2 bits, and XORs. The 32-bit f-function (see Fig. 1) used in the Feistel network takes a 16-bit subkey as a parameter and mixes it with the data using the functions S_0 and S_1. Both functions take two bytes as input and return a single output byte. They are defined as:

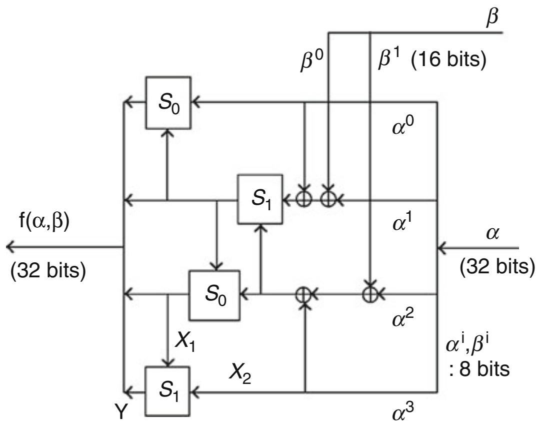
$$S_i(x, y) = [x + y + i(\bmod 2^8)] \lll 2, \qquad i \in \{0, 1\}.$$

Another structural difference with DES is the additional layers inserted before the first and after the last round of FEAL. In these layers, an extra 64-bit subkey is mixed with the data and the left half of the data is XORed with the right half.

While most of its components are far simpler, FEAL has a more complex *key schedule* (►Block Cipher) than DES. The secret key is expanded in a Feistel-like ladder network using a nonlinear f_k-function similar to the f-function mentioned above.

FEAL has been studied by many researchers, and various interesting techniques were developed to analyze its security. The next paragraph gives a short overview of the most important developments.

The first vulnerabilities in the original four-round version, FEAL-4, were discovered by den Boer [4] in 1988. He developed an adaptively chosen plaintext attack which was later improved by Murphy [8]. The improved attack required only 20 chosen plaintexts and at that time recovered the key in less than 4 h. At the Securicom conference in 1989, Biham and Shamir demonstrated a chosen plaintext attack on FEAL-8, the eight-round version of the cipher. The attack, which would turn out to be a direct application of ►differential cryptanalysis, is mentioned in different papers [1, 5, 7], but its details were only published in 1991 [2]. In 1990, Gilbert and Chassé [5] proposed a statistical chosen plaintext attack on FEAL-8. A year later, Tardy-Corfdir and Gilbert [10] presented a *known* plaintext attack on FEAL-4 and FEAL-6. The first attack had many ideas in common with differential cryptanalysis, which was being developed around the same time; the second attack contained elements which would later be used in ►linear cryptanalysis. In 1992, a first variant of this linear cryptanalysis was introduced by Matsui and Yamagishi [6]. The new attack could recover FEAL-4 keys in 6 minutes using only five known plaintexts. In 1993, Biham and Shamir [3] developed a more efficient differential attack against FEAL-8 deriving the secret key in 2 min on a PC given 128 chosen plaintexts.



FEAL. Fig. 1 FEAL's f-function

The development of these new techniques forced the designers of FEAL to considerably increase the number of rounds, sacrificing its original efficiency.

Recommended Reading

1. Biham E, Shamir A (1991) Differential cryptanalysis of DES-like cryptosystems. In: Menezes A, Vanstone SA (eds) Advances in cryptology – CRYPTO'90. Lecture notes in computer science, vol 537. Springer, Berlin, pp 2–21
2. Biham E, Shamir A (1991) Differential cryptanalysis of Feal and N-hash. In: Davies DW (ed) Advances in cryptology – EUROCRYPT'91. Lecture notes in computer science, vol 547. Springer, Berlin, pp 1–16

3. Biham E, Shamir A (1993) Differential cryptanalysis of the data encryption standard. Springer, Berlin
4. den Boer B (1988) Cryptanalysis of FEAL. In: Günther CG (ed) Advances in cryptology – EUROCRYPT'88. Lecture notes in computer science, vol 330. Springer, Berlin, pp 293–299
5. Gilbert H, Chassé G (1991) A statistical attack of the FEAL-8 cryptosystem. In: Menezes A, Vanstone SA (eds) Advances in cryptology – CRYPTO'90. Lecture notes in computer science, vol 537. Springer, Berlin, pp 22–33
6. Matsui M, Yamagishi A (1993) A new method for known plaintext attack of FEAL cipher. In: Rueppel RA (ed) Advances in cryptology – EUROCRYPT'92. Lecture notes in computer science, vol 658. Springer, Berlin, pp 81–91
7. Miyaguchi S (1990) The FEAL-8 cryptosystem and a call for attack. In: Brassard G (ed) Advances in cryptology – CRYPTO'89. Lecture notes in computer science, vol 435. Springer, Berlin, pp 624–627
8. Murphy S (1990) The cryptanalysis of FEAL-4 with 20 chosen plaintexts. J Cryptol 2(3):145–154
9. Shimizu A, Miyaguchi S (1988) Fast data encipherment algorithm FEAL. In: Chaum D, Price WL (eds) Advances in cryptology – EUROCRYPT'87. Lecture notes in computer science, vol 304. Springer, Berlin, pp 267–278
10. Tardy-Corfdir A, Gilbert H (1992) A known plaintext attack of FEAL-4 and FEAL-6. In: Feigenbaum J (ed) Advances in cryptology – CRYPTO'91. Lecture notes in computer science, vol 576. Springer, Berlin, pp 172–181

Feige–Fiat–Shamir Signature Scheme

►Fiat–Shamir Identification Protocol and the Feige–Fiat–Shamir Signature Scheme

Feistel Cipher

Alex Biryukov
FDEF, Campus Limpertsberg, University of Luxembourg, Luxembourg

Related Concepts

►Block Ciphers

Definition

One popular class of the modern iterative ►blockciphers is the *Feistel* ciphers (named so after Horst Feistel – cryptanalyst who worked with the IBM crypto group in the early 1970s). The round of a Feistel cipher uses the product of two involutions (a function G is called an involution if it is its own inverse: $G(G(x)) = x$) in order to achieve the very comfortable similarity of encryption and decryption processes.

Theory

Given an n-bit block, a Feistel round function divides it into two halves L (left) and R (right). Then some function $F(R, k)$ is applied to the right half and the result is XORed with the left half (this is the first involution):

$$(L, R) \to (R, L \oplus F(R, k)) .$$

Here k is the round subkey produced by the key scheduling algorithm; it may vary from round to round. Then the halves are swapped (the second involution) and the process is repeated. Another convenience in this construction is that it is always a permutation, and thus is invertible no matter what function F is used and thus a designer may now concentrate on the security properties of this function. Many modern ciphers are designed as Feistel ciphers. One prominent example of a Feistel cipher is the ►Data Encryption Standard.

Note that the division into halves in a Feistel cipher can be replaced by division into quarters, octets, etc. Such ciphers are called *generalized Feistel ciphers*. Several ciphers are of this type, for example the ►CAST family of ciphers or the cipher ►Skipjack.

Fermat Primality Test

Moses Liskov
Department of Computer Science, The College of William and Mary, Williamsburg, VA, USA

Synonyms

Fermat test

Related Concepts

►Miller–Rabin Probabilistic Primality Test; ►Modular Arithmetic; ►Primality Test; ►Prime Number

Definition

The Fermat primality test is a test to determine if a number is a ►prime number, using ►Fermat's little theorem.

Background

Fermat was not concerned with efficient algorithms for testing primality, but Fermat's little theorem (which states that $a^{p-1} \equiv 1 \bmod p$ whenever p is prime and a is relatively prime to p) gives a necessary condition for primality, thus the name.

Theory

Fermat's little theorem is not true for composite numbers in general, so it is an excellent tool to use to test for the compositeness of a number. The test consists of checking whether $a^{n-1} \equiv 1 \bmod n$ is satisfied for some a relatively prime to n. If not, it is certain that n is not prime.

Unfortunately, if $a^{n-1} \equiv 1 \bmod n$ does hold, it may be that n is composite. When this happens, n is said to be a ►"pseudoprime" for the base a. By picking many random a, some false positives can be avoided. The procedure is as follows, on input n:

```
1.  For $i = 1$ to $s$ do:
    (a) Pick a random $a$ such that $2 \le a \le n-1$ and $\gcd(a, n) = 1$
    (b) If $a^{n-1} \not\equiv 1 \bmod n$, return COMPOSITE
2.  Return PRIME
```

Here, s is a parameter. The running time of the Fermat primality test is $\Theta(s)$ modular exponentiations. For most composite numbers, the chance of passing these s rounds with different values for a without being detected as composite is $1/2^s$. However, there are some composite numbers, called Carmichael numbers, which pass the Fermat test for *every* base a and the smallest is $561 = 3 \cdot 11 \cdot 17$. Since 80 is a multiple of $3-1, 11-1$, and $16-1$, it holds that $a^{80} \equiv 1 \bmod 561$ for all a relatively prime to 561, so in particular, $a^{560} = (a^{80})^7$ is always congruent to 1 modulo 561. Carmichael numbers are relatively rare; asymptotically, if $C(n)$ is the number of Carmichael numbers less than n, then $n^{2/7} < C(n) < n^{1-\frac{\ln\ln\ln n}{\ln\ln n}}$.

Because of these exceptions, the Fermat primality test is usually discarded in favor of a more robust test such as the ►Miller–Rabin probabilistic primality test. However, PGP uses the Fermat primality test.

Recommended Reading

1. Conway JH, Guy RK (1996) The book of numbers. Springer, New York
2. Cormen TH, Leiserson CE, Rivest RL, Stein C (2001) Introduction to algorithms. MIT Press, Cambridge

Fermat Test

►Fermat Primality Test

Fermat's Little Theorem

Moses Liskov
Department of Computer Science, The College of William and Mary, Williamsburg, VA, USA

Related Concepts

►Euler's Totient Function; ►Fermat Primality Test; ►Modular Arithmetic; ►Number Theory; ►Prime Number

Definition

Fermat's little theorem states that if p is a prime number and a is any number not divisible by p, then $a^{p-1} \equiv 1 \bmod p$.

Background

Pierre de Fermat (1601–1665) was one of the most renowned mathematicians in history. He focused much of his work on Number Theory, though he made great contributions to many other areas of mathematics. Fermat's famous "last theorem" was a remark Fermat made in a margin of a book, for which he claimed to have a proof but the margin was too small to write it down, and remained an open problem for over three centuries.

Fermat's little theorem was first stated in a letter to Frénicle de Bessy in 1640, though it was not called "Fermat's little theorem" until the twentieth century.

Theory

The proof follows easily from the following observations. Consider the product $(a)(2a)(3a)\ldots((p-1)a)$, then on the one hand we can write this as $(p-1)!a^{p-1}$. On the other hand, the list of terms in the product, modulo p, is a complete list of values between 1 and $p-1$, since no two terms in the list are equivalent modulo p: if $na = ma \bmod p$ where $n > m$ then p divides $n-m$ but $0 < n-m < p-1$, which is a contradiction. Thus, this product can also be written as $(p-1)! \bmod p$. Thus, $(p-1)!a^{p-1} \equiv (p-1)! \bmod p$, and $a^{p-1} \equiv 1 \bmod p$, because $(p-1)!$ is not divisible by p.

Euler generalized Fermat's little theorem, which applies only to $(\mathbb{Z}/p\mathbb{Z})^*$, to any finite group. Fermat's little theorem is important to cryptography in that it gives rise to the ►Fermat primality test, a method for testing primality.

Recommended Reading

1. Conway JH, Guy RK (1996) The book of numbers. Springer, New York
2. Hardy GH, Wright EM (2008) An introduction to the theory of numbers. Oxford University Press, New York

Fiat–Shamir Identification Protocol and the Feige–Fiat–Shamir Signature Scheme

Yvo Desmedt
Department of Computer Science, University College London, London, UK

Related Concepts

►Digital Signatures; ►Feige–Fiat–Shamir Signature Scheme; ►Interactive Proof; ►Random Oracle Model; ►Zero-knowledge

Definition

Fiat–Shamir proposed the use of zero-knowledge interactive proofs for entity authentication and using their Fiat–Shamir trick to generate digital signatures.

Theory

Introduction

There are several variants of the *Fiat–Shamir identification protocol*. One way to classify these is based on the number of secrets. In the basic one [6], each prover knows only one secret. Another, is to distinguish between *identity-based* and *public key*–based ones. In both, a trusted center made public $n = p \cdot q$ such that p and q are secret primes only known to the center.

In the *identity-based* system [3, pp. 152] (see also [2, 6, 7]), a trusted center gives each user a *secret key*, partially based on biometrics. In particular, to receive an *identity* from the trusted center, Alice goes to the center. There her fingerprints, other biometrics information is collected and her identity verified. I is the string which contains: Alice's identity (name), Alice's biometrics, and other information to identify Alice uniquely. The center chooses k small j_i $(1 \le i \le k)$ such that $x_i := f(I, j_i)$ are quadratic residues mod$n$, where f is a public function. The center calculates as secrets the smallest $s_i := (\sqrt{x_i})^{-1}$ mod n and gives these secrets to Alice (write these in Alice's smart card).

In the public key–based system, Alice chooses her secrets and publishes a public key. In particular, Alice chooses uniformly random $s_i \in_R Z_n$ and computes $x_i := s_i^{-2}$ mod n. These x_i with Alice's identity is published in an authentic public directory.

Preliminary

In the *identity-based* scheme, when Bob wants to verify Alice's *identity*, Bob asks Alice's identity I together with the j_i, and Bob checks Alice's biometrics. If correct, Bob then calculates $x_i := f_i(I, j_i)$. Else Bob halts.

In the *public key*–based scheme, Alice reveals her *identity*, and Bob finds her public key in the public key directory.

The Protocol

Repeat the protocol t times:

Step 1 Alice chooses uniformly random r and computes $z := r^2$ mod n and sends t to Bob.
Step 2 Bob sends Alice bits e_i $(1 \le i \le k)$.
Step 3 Alice sends $\alpha := r * \prod_{e_i=1} s_i^{e_i}$ mod n.
Step 4 Bob verifies that $\alpha^2 * \prod_{e_i=1} x_i = z$.

If all the verifications are correct, Bob accepts.

Parameters

The size of t and k as functions of n are chosen depending whether one uses Fiat–Shamir as an identification protocol or as a *zero-knowledge interactive proof*. For more details see [1].

Feige–Fiat–Shamir Signature Scheme

The Fiat–Shamir scheme can be used to *digitally sign*. This is called the *Feige–Fiat–Shamir signature*. To sign a message m, just use a secure *hash function* h to compute e_i in the above protocol as follows:

$$e_i := h(z, m) \tag{1}$$

where z is as above. Use the Fiat–Shamir protocol and publish as signature: $(\mathbf{e}, \alpha)$ where $\mathbf{e} = (e_1, \ldots, e_k)$ and α is as above. The use of a hash function to compute $\mathbf{e}$ is often called the Fiat–Shamir trick.

To verify the signature, the verifier computes z' as in Step 4. Then the verifier uses this z' to compute $\mathbf{e}'$ using Eq. 1. If $\mathbf{e}' = \mathbf{e}$, the verifier accepts the signature of m.

The Fiat–Shamir trick provides secure digital signature schemes under the *random oracle model* [5]. Unfortunately, the random oracle model is a questionable proof technique, and a deeper study of this impact on the Fiat–Shamir can be found in [4].

Recommended Reading

1. Burmester MVD, Desmedt YG (1989) Remarks on the soundness of proofs. Electron Lett 25(22):1509–1511
2. Fiat A, Shamir A (1987) How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko A (ed) Advances in cryptology, Proc. of Crypto '86, Santa Barbara, CA, August 11–15 (Lecture notes in computer science 263), Springer-Verlag, Heidelberg, pp 186–194
3. Fiat A, Shamir A (1987) Unforgeable proofs of identity. In: Securicom 87, March 4–6, 1987. Paris, France, pp 147–153

4. Goldwasser S, Kalai YT (2003) On the (in)security of the at-shamir paradigm. In: 44th symposium on foundations of computer science (FOCS 2003), Proceedings, 11–14 October 2003, Cambridge, MA. IEEE Computer Society, pp 102–113
5. Pointcheval D, Stern J (1996) Security proofs for signature schemes. In: Maurer U (ed) Advances in cryptology | Eurocrypt '96, Proceedings, Zaragoza, Spain, May 12–16 (Lecture notes in computer science 1070), Springer-Verlag, Heidelberg, pp 387–398
6. Shamir A (1986) Interactive identification, March 23–29, 1986. In: Presented at the workshop on algorithms, randomness and complexity, Centre International de Rencontres Mathématiques (CIRM), Luminy (Marseille), France
7. Shamir A (1986) The search for provably secure identification schemes. In: Proceedings of the international congress of mathematicians, August 3–11, 1986. Berkeley, CA, pp 1488–1495

Field

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

Related Concepts

►Finite Field; ►Group; ►Ring

Definition

A field is a ►ring with a commutative multiplication operation and a well-defined multiplicative inverse (i.e., a "division" operation).

Theory

A *field* $F = (S, +, \times)$ is a ►ring that has a multiplicative identity (denoted 1) and satisfies two additional properties:

- **Commutativity of** $\times$: For all $x, y \in S$, $x \times y = y \times x$.
- **Multiplicative inverse**: For all $x \in S$ such that $x \neq 0$, there exists a *multiplicative inverse* z such that $x \times z = z \times x = 1$.

Thus, a field is a commutative ring with a multiplicative identity where every element except the additive identity has a multiplicative inverse, so that division is defined in the usual way (i.e., the quotient y/x is defined for all $x \neq 0$ and y as $y \times z$, where z is the multiplicative inverse of x).

The *characteristic* of a field is the least positive integer k such that for all $x \in S$, $kx = 0$, if such a k exists; it is defined as 0 otherwise. A ►finite field is field with a finite number of elements; the characteristic of a finite field is always a ►prime number. The rational numbers **Q** are an example of an infinite field under ordinary addition and multiplication.

Applications

Fields, particularly finite fields, are widely employed in both symmetric and asymmetric cryptography.

File System Permissions

►Permissions

Filter Generator

Anne Canteaut
Project-Team SECRET, INRIA Paris-Rocquencourt, Le Chesnay, France

Related Concepts

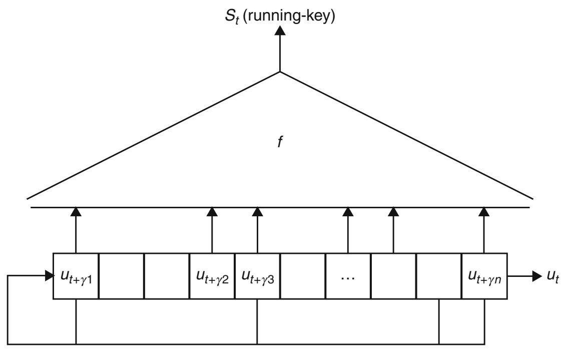
►Boolean Functions; ►Combination Generator; ►Linear Feedback Shift Register; ►Stream Cipher

Definition

A *filter generator* is a ►running-key generator for ►stream cipher applications. It consists of a single ►linear feedback shift register (LFSR) which is filtered by a nonlinear function. More precisely, the output sequence of a filter generator corresponds to the output of a nonlinear function whose inputs are taken from some stages of the LFSR. If $(u_t)_{t \geq 0}$ denotes the sequence generated by the LFSR, the output sequence $(s_t)_{t \geq 0}$ of the filter generator is given by

$$s_t = f(u_{t+\gamma_1}, u_{t+\gamma_2}, \ldots, u_{t+\gamma_n}), \qquad \forall t \geq 0$$

where f is a function of n variables, n is less than or equal to the LFSR length, and $(\gamma_i)_{1 \leq i \leq n}$ is a decreasing sequence of nonnegative integers called the *tapping sequence*.



Theory

In order to obtain a keystream sequence having good statistical properties, the filtering function f should be *balanced* (i.e., its output should be uniformly distributed), and the feedback polynomial of the LFSR should be chosen to be a primitive polynomial (►linear feedback shift register for more details).

In a filter generator, the LFSR feedback polynomial, the filtering function, and the tapping sequence are usually publicly known. The secret parameter is the initial state of the LFSR which is derived from the secret key of the cipher by a key-loading algorithm. Therefore, most attacks on filter generators consist in recovering the LFSR initial state from the knowledge of some digits of the sequence produced by the generator (in a ►known plaintext attack) or of some digits of the ciphertext sequence (in a ►ciphertext only attack). The attack presented in [12] enables to construct an equivalent keystream generator from the knowledge of a large segment of the ciphertext sequence when the LFSR feedback polynomial is the only known parameter (i.e., when the filtering function, the tapping sequence, and the initial state are kept secret).

Any filter generator is equivalent to a particular ►combination generator, in the sense that both generators produce the same output sequence. An equivalent combination generator consists of n copies of the LFSR used in the filter generator with shifted initial states; the combining function corresponds to the filtering function.

Statistical properties of the output sequence. The output sequence $\mathbf{s}$ of a filter generator is a linear recurring sequence. Its linear complexity, $\Lambda(\mathbf{s})$, is related to the LFSR length and to the *algebraic degree* of the filtering function f (the algebraic degree of a Boolean function is the highest number of terms occurring in a monomial of its algebraic normal form). For a binary LFSR with a primitive feedback polynomial, we have

$$\Lambda(\mathbf{s}) \leq \sum_{i=0}^{d} \binom{L}{i}$$

where L denotes the LFSR length and d denotes the algebraic degree of f [7, 9]. The period of $\mathbf{s}$ divides $2^L - 1$. Moreover, if L is a large prime, $\Lambda(\mathbf{s})$ is at least $\binom{L}{d}$ for most filtering functions with algebraic degree d (see [11]).

Thus, to achieve a high linear complexity, the LFSR length L and the algebraic degree of the filtering function should be large enough. More precisely, the keystream length available to an attacker should always be much smaller than $\binom{L}{\deg(f)}$.

Known attacks and related design criteria. Filter generators are vulnerable to ►fast correlation attacks because their output sequence is correlated to some linear function of the stages of the constituent LFSR (►fast correlation attack for details). In order to make the fast correlation attacks computationally infeasible, the filtering function should have a high nonlinearity. Moreover, it should have many nonzero Walsh coefficients.

Distinguishing attacks exploiting sparse multiples of the LFSR feedback polynomial can also be mounted, as described in [4] and [10]; their complexities involve the degree of sparse multiples of the LFSR feedback polynomial and the Walsh coefficients of the filtering function. In particular, the LFSR feedback polynomial should not be sparse.

Another attack on any filter generator is the generalized ►inversion attack. It highly depends on the *memory size* of the generator, which corresponds to the largest spacing between two taps, i.e., $M = \gamma_1 - \gamma_n$. To make this attack infeasible, the tapping sequence should be such that the memory size is large and preferably close to its maximum possible value, $L - 1$, where L is the LFSR length. Moreover, when the greatest common divisor of all spacing between the taps, $\gcd(\gamma_i - \gamma_{i+1})$, is large, the effective memory size can be reduced by a decimation technique (►inversion attack). Then, the greatest common divisor of all $(\gamma_i - \gamma_{i+1})$ should be equal to 1.

The choice of the tapping sequence also conditions the resistance to the so-called *conditional correlation attacks* [1, 5, 8]. The basic idea of these particular correlation attack is that some information on the LFSR sequence may leak when some patterns appear in the keystream sequence. Actually, the keystream bits s_t and $s_{t+\tau}$, with $\tau \geq 1$, respectively, depend on the LFSR-output bits $u_{t+\gamma_1}, \ldots, u_{t+\gamma_n}$ and $u_{t+\gamma_1+\tau}, \ldots, u_{t+\gamma_n+\tau}$. Therefore, the pair $(s_t, s_{t+\tau})$ only depends on $M - I(\tau)$ bits of the LFSR sequence, where M is the memory size and $I(\tau)$ is the size of the intersection between $\{\gamma_i, 1 \leq i \leq n\}$ and $\{\gamma_i + \tau, 1 \leq i \leq n\}$, i.e., the number of pairs (i, j) with $i < j$ such that $\gamma_i - \gamma_j = \tau$. It is then clear that a given observation of $(s_t, s_{t+\tau})$ may provide some information on the $(M - I(\tau))$ involved bits of the LFSR sequence when $I(\tau)$ is large. Thus, $I(\tau)$ should be as small as possible for all values of $\tau \geq 1$. It can be proved that the lowest possible value of $\max_{\tau\geq 1} I(\tau)$ is 1, and it is achieved when the tapping sequence is a *full positive difference set*, i.e., when all differences $\gamma_i - \gamma_j$, $i < j$ are distinct. Such a tapping sequence of n integers only exists if the LFSR length exceeds $n(n-1)/2$ (see [5]). More details of this attack and on the related security criteria for the filtering function can be found in [6].

Advanced algebraic techniques, like Gröbner bases, also provide powerful ►known plaintext attacks on filter generator, called *algebraic attacks* [3]. Any keystream bit can be expressed as a function of the L initial bits of the LFSR. Thus, the knowledge of any N keystream bits lead to an algebraic system of N equations of L variables. The degree of these equations correspond to the algebraic degree of the filtering function. But, efficient Gröbner bases techniques enable to substantially lower the degree of the equations by multiplying them by well-chosen multivariate polynomials. Then, it may be possible to recover the LFSR initial state by solving the algebraic system even if the filtering function has a high degree (see [2] for a survey). The related security criterion is that the filtering function must have a high ►algebraic immunity. More sophisticated attacks exploiting some algebraic properties of the filter generator includes ►cube attacks and several related distinguishing attacks.

Recommended Reading

1. Anderson RJ (1995) Searching for the optimum correlation attack. In: Fast Software Encryption 1994. Lecture notes in computer science, vol 1008. Springer, Heidelberg, pp 137–143
2. Canteaut A (2006) Open problems related to algebraic attacks on stream ciphers. In: Coding and cryptography – WCC 2005. Lecture notes in computer science, vol 3969. Springer, Heidelberg pp 120–134
3. Courtois NT, Meier W (2003) Algebraic attacks on stream ciphers with linear feedback. In: Advances in cryptology - EUROCRYPT 2003. Lecture notes in computer science, vol 2656. Springer, Heidelberg, pp 345–359
4. Englund H, Johansson T (2005) A new simple technique to attack filter generators and related ciphers. In: Selected areas in cryptography – SAC 2004. Lecture notes in computer science, vol 3357. Springer, Heidelberg, pp 39–53
5. Golić JDj (1996) On the security of nonlinear filter generators. In: Fast Software Encryption 1996. Lecture notes in computer science, vol 1039. Springer, Heidelberg, pp 173–188
6. Gouget A, Sibert H () Revisiting correlation-immunity in filter generators. In: Selected areas in cryptography – SAC 2007. Lecture notes in computer science, vol 4876. Springer, Heidelberg, pp 378–394
7. Key EL (1976) An analysis of the structure and complexity of nonlinear binary sequence generators. IEEE Trans Inform Theory 22:732–736
8. Lee S, Chee S, Park S, Park S. Conditional correlation attack on nonlinear filter generators. In: Advances in cryptology – ASIACRYPT'96. Lecture notes in computer science, vol 1163. Springer, Heidelberg, pp 360–367
9. Massey JL (2001) The ubiquity of Reed-Muller codes. In: Applied algebra, algebraic algorithms and error-correcting codes – AAECC-14. Lecture notes in computer science, vol 2227. Springer, Heidelberg, pp 1–12
10. Molland H, Helleseth T (2004) An improved correlation attack against irregular clocked and filtered generator. In: Advances in cryptology – CRYPTO 2004. Lecture notes in computer science, vol 3152. Springer, Heidelberg, pp 373–389
11. Rueppel RA (1986) Analysis and design of stream ciphers. Springer, New York
12. Siegenthaler T (1985) Decrypting a class of stream ciphers using ciphertext only. IEEE Trans Comput C-34(1):81–84

Fingerprint

Ruggero Donida Labati, Fabio Scotti
Dipartimento di Tecnologie dell'Informazione,
Universita' degli Studi di Milano, Crema (CR), Italy

Related Concepts

►Biometric Identification

Definition

Fingerprints are reproductions of the surface pattern of the fingertip epidermis. This pattern is a characteristic sequence of interleaved ridges and valleys, usually considered as unique for each individual.

Background

Fingerprints are the most used and known biometric traits. Biometric systems based on the fingerprint trait estimate the identity of an individual by extracting and comparing information related to the characteristics of the ridge pattern.

Fingerprint recognition systems are used in different contexts [1]. Important applications of the fingerprint trait are the forensic investigation, and the public sector applications (e.g., border controls, police archives, national archives, and biometric documents). Also in the private sector, fingerprint biometric systems are commonly deployed to control the accesses to critical areas, consoles, data and electronic devices (e.g., personal computers, PDA, and mobile phones). Emerging sectors for the application of these technologies are e-government, e-commerce, and e-banking.

Fingerprint recognition is the most mature biometric technology. In fact, fingerprint was introduced as a method for person identification before 1900, and the first automatic fingerprint recognition systems was introduced in the 1970s.

Fingerprint is a biometric trait with high permanence and distinctiveness. The fingertip ridge structure is fully formed at about 7 months of fetus development, and this pattern configuration does not change for all the life unless serious accidents or diseases. Usually, cuts and bruises can only temporarily modify the fingerprint pattern.

In general, fingerprints are a part of an individual's phenotype and are different for each individual. Also the fingerprints of the same person are different, and even in the case of the fingerprints of identical twins are not equal. The social acceptability of the use of the fingerprint biometric trait can be considered as limited, since people perceive this biometric trait as related to police and investigative activities.

Theory

Fingerprint Images

Three different classes of fingerprint images can be identified: latent fingerprints, inked fingerprints, and live-scan fingerprints.

Latent fingerprints are very important in forensics. This kind of fingerprints is produced by the transfer of the film of moisture and grease that is present on the surface of the finger when contacts to objects occur. Usually, latent fingerprint are not visible to naked eye and forensic investigators use proper substances for enhancing the visibility of the ridge pattern.

Inked fingerprints are typically obtained with the following procedure. The user's finger is spread with black/blue ink and then rolled on a paper card, secondly the card is converted into a digital form by means of a high-definition paper-scanner or by using a high-quality CCD camera.

Live-scan fingerprints are obtained by impressing a finger on the acquisition surface of a device. It is possible to distinguish three different types of live-scan sensor technologies: optical sensors; solid state sensors; ultrasound sensors. Figure 1 shows an example of images obtained by the different fingerprint acquisition methods.

Only good quality images have to be stored in order to achieve subsequently accurate biometric recognitions by automatic systems. For example, the Federal Bureau of Investigation (FBI) of the USA defined a set of main parameters characterizing the acquisition of a digital fingerprint image encompassing minimum resolution, minimum size of the captured area, minimum number of pixels, maximum geometry distortion of the acquisition device, minimum number of gray-levels, maximum gray-level uniformity, minimum spatial frequency response of the device, and minimum signal to noise ratio.

In many law enforcement and government applications when Automatic Fingerprint Identification Systems (AFIS) are involved, the size of the fingerprint database is typically very large. For example, the FBI fingerprint card archive contains over 200 million of identities. In such cases, compressed formats are adopted. One of the most used compression algorithm had been proposed by the FBI and it is based on the Wavelet Scalar Quantization (WSQ) technique.

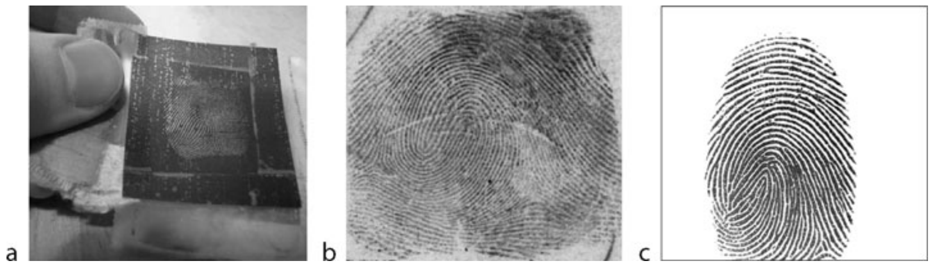
Fingerprint Analysis

The analysis of the ridge details can be performed with three different levels of accuracy.

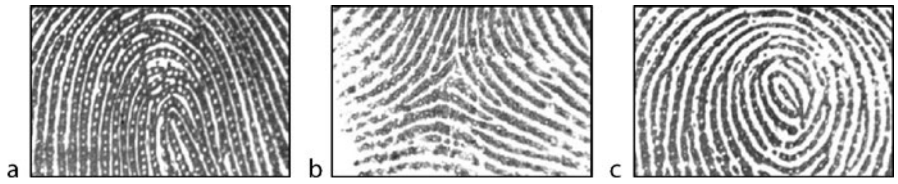
- Level 1: the overall global ridge flow pattern is considered.
- Level 2: the analysis is based on distinctive points of the ridges, called minutiae points.
- Level 3: ultra-thin details, such as pores and local peculiarities of the ridge edges, are studied.

Examples of characteristics analyzed at Level 1 are the *local ridge orientation*, the *local ridge frequency*, the *singular regions*, and the *ridge count*. The local ridge orientation is estimated as the angle of the ridges with respect to the horizontal axis. The ridge orientation map of the fingerprint can be computed by estimating the local ride orientation in areas centered in each pixel of the fingerprint images. In the literature, most of the methods for the computation of the ridge orientation map are based on the analysis of the gradient of the fingerprint image [2].

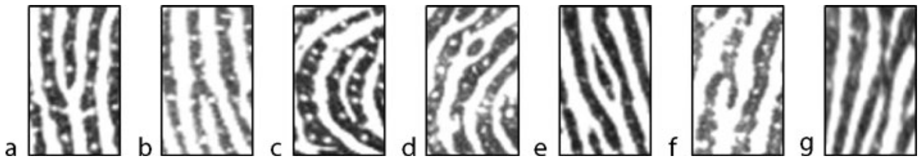
The local ridge frequency is the number of ridges per unit length along a segment that is orthogonal to the ridge orientation. The ridge frequency map represents the local ridge frequency computed in each pixel of the fingerprint image.



Fingerprint. Fig. 1 Examples of fingerprint images: (**a**) latent; (**b**) rolled and inked; (**c**) live-scan



Fingerprint. Fig. 2 Examples of singular regions: (**a**) loop; (**b**) delta; (**c**) whorl



Fingerprint. Fig. 3 Examples of common minutiae types: (**a**) termination; (**b**) bifurcation; (**c**) lake; (**d**) point or island; (**d**) independent ridge; (**f**) spur; (**g**) crossover

An important Level 1 analysis consists of the estimation of the singular regions. A singular region represents an area of the finger with a distinctive shape of the ridges. Commonly, three different types of singular regions are considered: loop, delta, and whorl. The distinctive shapes of these regions are ∩, Δ, and O respectively (Fig. 2). In the literature, the majority of the methods for the estimation of the singular points uses the Poincaré technique [1] applied on the ridge orientation map. From the analysis of the singular regions, it is also possible to estimate a reference point in the fingerprint, which is called *core point*. Examples of other characteristics of Level 1 are the *ridge count* [3] and other global mapping information obtained by the application of Gabor filters on the input fingerprint images [4].

Level 2 analysis evaluates specific ridge discontinuities called *minutiae*. It is possible to distinguish many different classes of minutiae (Fig. 3), but most of the automatic biometric systems in the literature consider only terminations and bifurcations. Usually, the identification of the minutia can be achieved in four main steps [1]: (1) an adaptive binarization is applied in order to separate the ridges from the background in the fingerprint image; (2) a thinning operation is achieved in order to reduce the thickness of the ridges to one single pixel; (3) the coordinates of the minutiae are estimated by observing the specific local pattern of each single pixel of the ridges, typically in its eight-neighborhood; (4) a post-processing method is applied in order to reduce the number of false minutiae detected in the previous step.

A different approach for the minutiae identification processes directly gray-scale images without the binarization step and it based on the capability to follow the ridges in the input image by observing the local orientation [5]. Another important information related to each minutia is its direction, considered as the value of the ridge orientation map in the minutia coordinates.

An additional step that can be present in the fingerprint analysis is the *image enhancement*. Different algorithms are available [1], for example, pixel-wise enhancement, contextual filtering, and multi-resolution enhancement. One of the most famous enhancement method applies a set of Gabor filters to the fingerprint image according to the ridge orientation map and the ridge frequency map [6].

Level 3 analysis requires high-resolution acquisition devices (with at least 800 dpi) and it is not commonly applied in commercial systems. Standard features extracted at this level of analysis consist in the spatial coordinates of the pores.

Captured fingerprint images can have very different quality levels. In particular, low levels of the fingerprint quality can compromise the identity verification/recognition [7]. In order to control this factor, quality estimation methods are usually considered [8, 9]. Quality estimation is also useful to select unrecoverable image regions, to reduce the artifacts product by the enhancement algorithms, and to properly weight the extracted features according to the local quality level of the input fingerprint.

Fingerprint Matching

Fingerprint matching algorithms compute a similarity index called *match-score* of two fingerprints used in the verification/identification procedures. It is possible to divide the fingerprint matching algorithms in three different classes: correlation-based techniques; minutiae-based methods; and methods based on other features.

The *correlation-based* techniques computes the match-score between two fingerprint images. The images are scaled, translated, rotated, equalized, and then, the match-score is obtained as the correlation between the two images. This approach is rather basic and not commonly present in automated fingerprint recognition system.

The *minutiae-based* methods are the most studied and applied in the literature [10]. Most of the methods perform an alignment of the minutiae extracted from two fingerprint images by shift and rotate operations, and then the match-score is computed as the number of matched minutiae pairs divided by the mean number of minutiae in the two images. The alignment step is necessary for reducing problems related to difference of scale, rotation, translation, and non linear distortion caused by different pressures of the finger on the sensor. It is possible to distinguish global and local minutiae matching algorithms. Global algorithms consider all the minutiae points of the two fingerprint images and search the best match-score by using different alignment strategies. For example, global minutiae matching methods can be based on algebraic geometry, Hough transform, relaxation, energy minimization. Local algorithms consider sets of minutiae divided in sub-portions, for example by adopting auxiliary graph structures (e.g., Delaunay triangles [11]). Local minutiae matching methods can be based on triangulation, multiple registration strategies, and warping techniques.

Other fingerprint matching methods can use features extracted at different levels as support information for processing a match-score based on the minutiae set or directly process features extracted at Level 1 or Level 3 (e.g., the template creation technique processed at Level 1 called *Fingercode* [4]).

Fingerprint Classification and Indexing

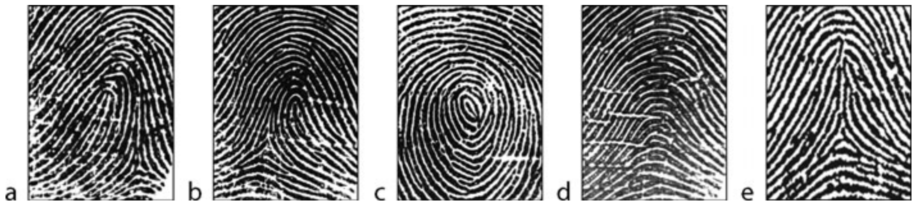
The identification procedure requires to compare the captured biometric template with all the templates stored in a database. In the case of very large databases, the computational time required for a complete full set of biometric comparisons can be unacceptable. A strategy used for reducing the required number of identity to be compared consists in the creation of partitions (called *bins*) containing only fingerprint that appertain to a defined class.

Usually, fingerprint are classified by the Galton–Henry classification scheme. This scheme is based on Level 1 features and is composed by 5 classes (arch, tented-arch, left loop, right loop, and whorl). Figure 4 shows an example of the considered classes. Fingerprint classification can be achieved by different approaches, such as neural networks classifiers, statistical methods, syntactic methods, rule-based methods, support vector machines [12]. Further strategies can be applied for reducing the number of identity comparisons such as the use of a subclassification [13] and the computation of continuous indexes related to different fingerprint features [11].

Security and Privacy in Fingerprint Recognition Systems

All hardware and software modules composing a fingerprint recognition system can be subject to attacks. In the literature, the majority of the techniques focuses on the injection of fingerprint data (e.g., fake fingers, injection of fingerprint data in the communication channel or in the storage system) and the replacement of software modules with a modified/malicious version in order to force/avoid the final match-score.

The former approach of attacks can be countered by vitality controls [14]. It is possible to divide these methods in two categories: methods based on involuntary captured information by the biometric scanner, and methods that measure a voluntary response of the user. The first category evaluates directly physical characteristics (e.g., heartbeat, odor, arterial oxygen saturation of hemoglobin, pulse presence) or characteristics extracted from one single fingerprint images or frame sequences (e.g., fine movements of the fingertip surface, involuntary challenge-response, valley noise, pores presence). The second approach analyzes the voluntary behavior of the user, information related to other biometric traits as in multibiometrics [15], additional data provided by the user (e.g.,



Fingerprint. Fig. 4 Galton–Henry classification scheme: (**a**) left loop; (**b**) right loop; (**c**) whorl; (**d**) arch; (**e**) tented arch password, smart-card, and voluntary challenge-response) or surveillance.

The protection of the user privacy is an important issue to be considered when fingerprint recognition systems are deployed [16, 17]. Techniques based on biometric encryption are often considered to achieve the protection of the fingerprint template [18–21]. The most known methods are the following: salting techniques, systems that non-invertible transforms, key-binding biometric cryptosystem, key generating biometric cryptosystem, and homomorphic cryptosystems.

Technologies for decentralizing fingerprint recognition computation are also available with specific reference to the match-on-card systems, system-on-device and system-on-a-chip technologies. An important advantage offered by these technologies is that the user can keep the possession of her/his templates (mostly on a tamper resistant hardware). The main disadvantage of such approach is that the obtained performances in terms of accuracy and computational time tend to be inferior than the performances of dedicated and PC-based fingerprint systems.

Applications

The applications of fingerprint technology are heterogeneous and range from the public to private sector. In the market there are available fingerprint recognition systems with a great difference of sizes of sensors, costs, and accuracy [1, 22]. For example, there are systems integrated in electronic devices (e.g., PDA and mobile phones), on-card systems, systems based on a personal computers, and large distributed systems [23] such as the AFIS [24].

The main applicative contexts of fingerprint recognition systems are in forensics, governmental, and commercial sectors. In the forensic sector, the fingerprint trait is used for the identification of persons, the search of lost person, and general investigative activities. In the governmental sector, important applications are border controls, biometric documents (e.g., passports and IDs). Examples of applications in the commercial sector are authentication systems integrated to ATM, terminal login, access control for on-line services (e.g., e-commerce and e-banking applications), and the protection of sensible data (e.g., in personal computers, PDA, mobile phones, and storage devices) and access control to restricted areas.

Recommended Reading

1. Maltoni D, Maio D, Jain AK, Prabhakar S (2009) Handbook of fingerprint recognition, 2nd edn. Springer, New York
2. Hou Z, Yau W-Y, Wang Y (2010) Security and communication networks. John Wiley & Sons, Ltd
3. Lin W-C, Dubes RC (1983) A review of ridge counting in dermatoglyphics. Pattern Recognition 16(1):1–8
4. Jain A, Prabhakar S, Hong L, Pankanti S (2000) Filterbank-based fingerprint matching. IEEE Trans Image Proces 9(5):846–859
5. Maio D, Maltoni D (1997) Direct gray-scale minutiae detection in fingerprints. IEEE Trans Pattern Anal Mach Intell 19:27–40
6. Hong L, Wan Y, Jain A (1998) Fingerprint image enhancement: algorithm and performance evaluation. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(8):777–789
7. Gamassi M, Lazzaroni M, Misino M, Piuri V, Sana D, Scotti F (2005) Quality assessment of biometric systems: a comprehensive perspective based on accuracy and performance measurement. IEEE Trans Instrum Meas 54(4):1489–1496
8. Alonso-Fernandez F, Fierrez J, Ortega-Garcia J, Gonzalez-Rodriguez J, Fronthaler H, Kollreider K, Bigun J (2007) A comparative study of fingerprint image-quality estimation methods. IEEE Trans Inf Forensic Secur 2(4):734–743
9. Donida Labati R, Piuri V, Scotti F (2010) Neural-based quality measurement of fingerprint images in contactless biometric systems. In: The 2010 international joint conference on neural network (IJCNN), Barcelona, pp 1–8
10. Yager N, Amin A (2004) Fingerprint verification based on minutiae features: a review. Pattern Anal Appl 7:94–113
11. Liang X, Bishnu A, Asano T (2007) A robust fingerprint indexing scheme using minutia neighborhood structure and low-order delaunay triangles. IEEE Trans Inf Forensic Secur 2(4):721–733
12. Cappelli R, Maio D (2004) The state of the art in fingerprint classification. In: Ratha N, Bolle R (eds) Automatic fingerprint recognition systems. Springer, New York, pp 183–205
13. Drets GA, Liljenström HG (1999) Fingerprint sub-classification: a neural network approach. In: Intelligent biometric techniques in fingerprint and face recognition. CRC, Boca Raton, pp 107–134
14. Coli P, Marcialis G, Roli F (2007) Vitality detection from fingerprint images: a critical survey. In: Lee S-W, Li S (eds) Advances in biometrics. Lecture Notes in Computer Science, vol 4642. Springer, Berlin, pp 722–731
15. Azzini A, Marrara S, Sassi R, Scotti F (2008) A fuzzy approach to multimodal biometric continuous authentication. Fuzzy Optim Decis Mak 7:243–256
16. Cimato S, Gamassi M, Piuri V, Sassi R, Scotti F (2008) Privacy in biometrics. In: Biometrics: theory, methods, and applications. Wiley, New York
17. Cimato S, Gamassi M, Piuri V, Sassi R, Scotti F (2008) Privacy-aware biometrics: design and implementation of a multimodal verification system. In: Annual computer security applications conference (ACSAC), Anaheim, pp 130–139
18. Jain AK, Nandakumar K, Nagar A (2008) Biometric template security. EURASIP J Adv Signal Process 2008:1–17
19. Barni M, Bianchi T, Catalano D, Di Raimondo M, Donida Labati R, Failla P, Fiore D, Lazzeretti R, Piuri V, Piva A, Scotti F (2010) A privacy-compliant fingerprint recognition system based on homomorphic encryption and fingercode templates. In: Fourth IEEE international conference on biometrics: theory applications and systems (BTAS), Washington, DC, pp 1–7
20. Barni M, Bianchi T, Catalano D, Raimondo MD, Donida Labati R, Failla P, Fiore D, Lazzeretti R, Piuri V, Scotti F, Piva A (2010) Privacy preserving fingercode authentication. In: Proceedings of the 12th ACM workshop on multimedia and security. ACM, New York, pp 231–240
21. Bianchi T, Donida Labati R, Piuri V, Piva A, Scotti F, Turchi S (2010) Implementing fingercode-based identity matching in the

encrypted domain. In: 2010 IEEE workshop on biometric measurements and systems for security and medical applications (BIOMS), Taranto, pp 15–21

22. Piuri V, Scotti F (2008) Fingerprint biometrics via low-cost sensors and webcams. In: Second IEEE international conference on biometrics: theory, applications and systems (BTAS), Crystal City, pp 1–6
23. Gamassi M, Piuri V, Sana D, Scotti F, Scotti O (2006) Scalable distributed biometric system – advanced techniques for security and safety. IEEE Instrum Meas Mag 9(2):21–28
24. Komarinski P (2005) Automated fingerprint identification systems (AFIS). Elsevier Academic, Amsterdam

Fingerprint Authentication

►Biometric Authentication

Fingerprinting

Alexander Barg[1], Gregory Kabatiansky[2]
[1]Department of Electrical and Computer Engineering, University of Maryland, College Park, MD, USA
[2]Dobrushin Mathematical Lab, Institute for Information Transmission Problems RAS, Moscow, Russia

Related Concepts

►Broadcast Encryption; ►Traitor Tracing; ►Watermarking

Definition

Fingerprinting is a technique that aims at preventing unauthorized redistribution of digital content or "digital piracy." This assumes a multitude of scenarios under which identical or very close copies of a document v (software, images, or other digital media) are made available to a large number of users of the system by paid subscription. Fingerprinting consists in embedding a mark x (*a fingerprint*) in the document with the purpose of encoding the identity of the user. Fingerprints should not be easily detectable or removable; they must be also designed in a way that makes forgery difficult or expensive. In the area of content distribution, the functionality associated with the fingerprinting system is its resistance to a *collusion attack*. A group or *coalition* of users of the system is said to perform a collusion attack if they produce a copy of the document v with either an obfuscated (for instance, totally removed) fingerprint or a fingerprint that does not enable the distributor D to trace it back to any of the members of the coalition. A closely related concept of ►watermarking is aimed at tracing the owner of the contents and is not intended to fight collusion attacks.

Background

The first mention of fingerprinting in the literature dates back to 1983 [18]. Presently there are two main trends in fingerprinting, depending on the acceptance of the so-called *marking assumption*. Under this assumption, the document v and the fingerprint x are strings over some finite alphabet $\mathcal{Q}$ (a typical length of v is in the range of megabytes or greater, while the length of x is on the order of several kilobytes). The location of x in v is not known to the users nor is it assumed that the bits of x form a consecutive substring of v. However, for practical reasons, fingerprints occupy the same positions in all the copies of v. A pirate user or a coalition of t such users can detect some positions that belong to x. It is assumed that *changing any undetectable position of the user's copy of the document makes it useless* and obliterates its market value. This scenario applies for instance to fingerprinting licensed software. Fingerprinting under the marking assumption was introduced by Chor et al. who studied distribution of decryption keys used to access pay-per-view TV programs and similar applications [10]. The main tools of mathematical analysis of fingerprinting with the marking assumption are combinatorial-, information-, and coding-theoretic.

In the absence of the marking assumption, both the distributor and the users are allowed to add relatively small distortion to the original document. The main application of this scenario is *fingerprinting of digital media* such as image, video, audio, and speech content. In such applications, slight differences from the original version will not affect the quality of users' copies. It is assumed that both the signal and the fingerprint are real random variables that obey some probability distribution. Establishing resilience of the system against collusion attacks relies on probabilistic or information-theoretic analysis.

Theory

Fingerprinting of Digital Data

Let $\mathcal{M} = \{1, 2, \ldots, M\}$ be the set of users of a distribution system of fingerprinted data. It is assumed that each of the M users of the system is assigned a fingerprint given by an n-dimensional vector over a finite alphabet $\mathcal{Q}$. The design of the system involves the following components:

1. The assignment of fingerprints to the users is accomplished by a mapping $f_k : \mathcal{M} \to \mathcal{Q}^n$, where $k \in \mathcal{K}$ and $\mathcal{K}$ is the set of keys used to hide the encoding mapping from possible collusion attacks by the users.

2. The identification of the members of the pirate coalition relies on a mapping $\phi_k : \mathcal{Q}^n \to \mathcal{M} \cup \{\emptyset\}$ that is designed to enable the distributor to locate the pirates irrespective of the coalition's strategy.

Let $U = \{u_1, \ldots, u_t\} \subset \mathcal{M}$ denote the pirate coalition and let $f_k(U) = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$ be the set of the fingerprints of its members, where $\boldsymbol{x}_i = f(u_i)$ for $i = 1, \ldots, t$. The pirates launch an attack against the distributor D using a stochastic mapping defined by a probability distribution $V(\boldsymbol{y}|f_k(U))$. Both the distributor and the pirates optimize their strategies in an attempt to accomplish their goals. The pirates choose the mapping V that maximizes the error probability of identification by D, while the distributor constructs the fingerprinting code (f_k, ϕ_k) that ensures reliable identification for all coalition strategies V.

The fingerprinting problem of this kind was introduced in as a part of a broader area known as ▸traitor tracing and formalized in later works [10]. The main body of work that relates to the digital fingerprinting problem relies on the marking assumption mentioned in the introduction. Call coordinate i of the fingerprints *undetectable* for the coalition U if

$$x_{1i} = x_{2i} = \cdots = x_{ti}$$

and *detectable* otherwise. The marking assumption states that for any fingerprint $\boldsymbol{y}$ created by the coalition, $y_i = x_{1i} = x_{2i} = \cdots = x_{ti}$ in every undetectable coordinate i. Let $\mathcal{E}(U) = \{y\}$ be the set of all fingerprints y that can be created by the coalition U following the marking assumption, called the *envelope* of the coalition. This assumption implies that the coalition's strategy obeys the condition

$$V(\boldsymbol{y}|\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) > 0 \text{ only if } \boldsymbol{y} \in \mathcal{E}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t).$$

Only such *admissible* strategies are considered in the system design.

The definition of the envelope $\mathcal{E}$ depends on the exact description of the rule that the coalition follows to modify the detectable positions:

- The *narrow-sense envelope* is obtained if the coalition is restricted to use only a symbol from their assigned fingerprints in the detectable positions:
$$\mathcal{E}_N(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) = \{\boldsymbol{y} \in \mathcal{Q}^n | y_i \in \{x_{1i}, \ldots, x_{ti}\}, \forall i\}. \quad (1)$$
- The *wide-sense envelope* permits the coalition to use any symbol from the alphabet $\mathcal{Q}$ in the detectable positions:
$$\mathcal{E}_W(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t) = \{\boldsymbol{y} \in \mathcal{Q}^n | y_i = x_{1i}, \forall i \text{ undetectable}\}. \quad (2)$$

A fingerprinting code that has the tracing capability for the narrow-sense rule is said to possess a t-identifiable parent property (IPP). The study of such codes was undertaken in [1, 6, 7, 12]. The codes are designed to be able to always recover at least one member of the pirate coalition, i.e., assume a zero error probability of identification. It is easy to show that if the alphabet size $q = |\mathcal{Q}| \le t$, the maximum number of users of such a system is $M \le t$. For $q \ge t+1$ it is possible to construct codes with the t-IPP of size M such that $\log M = \Omega(n)$; see [1, 6]. Upper bounds on the size of such codes were obtained in [2, 8].

The wide-sense envelope reflects a more general problem statement under which the members of the coalition have more amplitude in creating an unregistered copy of the document, In particular, the wide-sense envelope of the coalition is larger than the narrow-sense one for all alphabets of size $q \ge 3$. Properties of fingerprinting codes resistant to the collusion attack in the wide sense were established for any alphabet size $q \ge 2$ [5, 9].

A randomized fingerprinting code is a random variable (F, Φ) taking values in the family $\{(f_k, \phi_k), k \in \mathcal{K}\}$ with probability $\pi(k)$ defined on the set of keys $\mathcal{K}$. The error probability of identification is defined as

$$e(U, F, \Phi, V) = \sum_{k \in \mathcal{K}} \pi(k) \sum_{y : \phi_K(y) \notin U} V(\boldsymbol{y}|f_K(U)),$$

where $f_K(U)$ is the set of fingerprints of the members of the coalition U. Let

$$e_{\max}(F, \Phi, V) = \max_{U \subset \mathcal{M}, |U| = t} e(U, F, \Phi, V)$$

be the maximum probability of error. A number $R > 0$ is an ϵ-achievable rate for q-ary t-fingerprinting if for every $\delta > 0$ and a sufficiently large n there exists a q-ary randomized code (F, Φ) of length $n = n(\delta)$ with rate

$$\frac{1}{n} \log_q M > R - \delta$$

such that $e_{\max}(F, \Phi, V) < \epsilon$ for every admissible strategy V of the coalition. The capacity of q-ary t fingerprinting $C_{t,q}$ is defined as the supremum of ϵ-achievable rates as ϵ becomes arbitrarily small.

The formulation of the fingerprinting problem as a communication problem was developed in [4, 5, 16]. Lower bounds of the form $C_{t,2} = \Omega(t^2)$ were established in [3, 11, 17]. One of these constructions also includes a simple identification algorithm of pirates [17]. The best estimate $C_{t,2} \ge 1/(2t^2 \ln 2)$ is due to [3]. Regarding small values of t it was shown that $C_{2,2} \ge 0.25, C_{3,2} \ge 1/12$ [4]. Upper bounds of the form

$$C_{2,2} \le 0.25, \; C_{3,2} \le 0.0975, \; C_{t,2} \le 1/(t^2 \ln 2)$$

were announced in [3, 14]. At the time of writing this entry, complete proofs of the upper bounds have not been published.

Generalizations of the marking assumption considered in the literature include settings in which members of the coalition follow the marking assumption in all but a certain small number of coordinates of the fingerprint or erase some coordinates of the fingerprint, replacing it by unreadable symbols [5, 9]. Another variation of the fingerprinting problem, called *asymmetric fingerprinting*, arises when it is assumed that not only coalitions of users but also the distributor may create unauthorized copies of the content [15]. The goal of asymmetric fingerprinting is to bar the distributor from issuing an unregistered copy, while at the same time allowing it to trace the source of such copy to the user whose mark it contains.

Fingerprinting of Analog Data

Another case of the problem that has also been studied in the literature relates to the fingerprinting of continuous data with applications to media fingerprinting. It is natural to assume that the original document $\boldsymbol{v}$ is represented by a vector in $\mathbb{R}^n$. The marked copy of the ith user is obtained by computing $\boldsymbol{x}_i = \boldsymbol{v} + \boldsymbol{w}_i$, where $\boldsymbol{w}_i$ is the ith fingerprint (another real vector).

The most common assumption in the literature is that the document is a random vector $(v_1, \ldots, v_n)$ whose coordinates are real random $N(0,1)$-variables (Gaussian random variables with zero mean and unit variance). Similarly, fingerprints $\boldsymbol{w}_i, i = 1, \ldots, M$ are chosen as Gaussian vectors or taken as linear combinations of orthogonal pseudo-noise vectors. By assumption, the deviation of $\boldsymbol{x}_i$ from $\boldsymbol{v}$ has to be bounded above uniformly for all i: $\|\boldsymbol{x}_i - \boldsymbol{v}\| \leq \delta_1\sqrt{n}$, where δ_1 is an appropriately chosen threshold. In the case of Gaussian fingerprints, it is assumed that the variance α^2 of their coordinates satisfies the inequality $\alpha < \delta_1$ (otherwise with large probability, marked copies will violate the bounded deviation assumption).

Collusion attacks in this context proceed by averaging the copies of the document belonging to the users in the coalition or by taking their convex combination and adding some Gaussian noise or by some similar general method. Let $\boldsymbol{y}$ be the fingerprint formed by a pirate coalition. Detection is performed by correlation analysis: the ith user is assumed guilty if $(\boldsymbol{y}, \boldsymbol{x}_i) \geq \delta_2\|\boldsymbol{y}\|$, where δ_2 is another threshold. It is proved that for Gaussian fingerprints, the minimum size of the coalition that succeeds in removing any fingerprint with probability bounded away from zero, must be proportional to $\sqrt{n/\log M}$ [13].

Recommended Reading

1. Alon N, Cohen G, Krivelevich M, Litsyn S (2003) Generalized hashing and parent-identifying codes. J Comb Theory Series A 104:207–215
2. Alon N, Stav U (2004) New bounds on parent-identifying codes: the case of multiple parents. Comb Probab Comput 13(6):795–807
3. Amiri E, Tardos G (2009) High rate fingerprinting codes and the fingerprinting capacity, Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2009). New York, pp 336–345
4. Anthapadmanabhan NP, Barg A, Dumer I (2008) On the fingerprinting capacity under the marking assumption. IEEE Trans Inform Theory 54(6):2678–2689
5. Barg A, Blakley GR, Kabatiansky G (2003) Digital fingerprinting codes: problem statements, constructions, identification of traitors. IEEE Trans Inform Theory 49(4):852–865
6. Barg A, Cohen G, Encheva S, Kabatiansky G, Zémor G (2001) A hypergraph approach to the identifying parent property: the case of multiple parents. SIAM J Discrete Math 14: 423–431
7. Blackburn SR (2003) Combinatorial schemes for protecting digital content. In: Wensley CD (ed) Surveys in combinatorics 2003. Cambridge University Press, Cambridge, pp 43–78
8. Blackburn SR (2003) An upper bound on the size of a code with the k-identifiable parent property. J Comb Theory Series A 102:179–185
9. Boneh D, Shaw J (1998) Collusion-secure fingerprinting for digital data. IEEE Trans Inform Theory 44(5):1897–1905
10. Chor B, Fiat A, Naor M (1994) Traitor tracing. In Advances in Cryptology – CRYPTO '94, Lecture Notes in Computer Science, vol. 839. Springer, Berlin, pp 480–491
11. Dumer I, Equal-weight fingerprinting codes. In: Xing C et al. (eds) Second International Workshop on Coding Theory and Cryptography, China, June 2009. Lecture Notes in Computer Science, vol. 5557. Springer, Berlin, pp 43–51
12. Hollmann HDL, van Lint JH, Linnartz J-P, Tolhuizen LMGM (1998) On codes with the identifiable parent property. J Comb Theory Series A 82(2):121–133
13. Kilian J, Leighton FT, Matheson LR, Shamoon TG, Tarjan RE, Zane F (1998) Resistance of digital watermarks to collusive attacks, 1998. Princeton Computer Science Technical Report TR-585-98, Princeton University, Princeton, New Jersey
14. Huang Y-W, Moulin P (2009) Saddle-point solution of the fingerprinting capacity game under the marking assumption. In Proceedings of the 2009 IEEE International Symposium on Information Theory, Seoul, Korea, June 28–July 3, 2009, pp 2256–2260
15. Pfitzmann B, Schunter M (1996) Asymmetric fingerprinting. In: Maurer U (ed) Advances in Cryptology – EUROCRYPT 96, Lecture Notes in Computer Science, vol 1070. Springer, Berlin, pp 84–95
16. Somekh-Baruch A, Merhav N (2005) On the capacity game of private fingerprinting systems under collusion attacks. IEEE Trans Inform Theory 51(3):884–899
17. Tardos G (2008) Optimal probabilistic fingerprint codes. J Assoc Comput Mach 55(2):24
18. Wagner N (1983) Fingerprinting. In Proceedings of the 1983 IEEE Symposium on Security and Privacy, Apr 1983, Oakland, California, pp 18–22

Finite Field

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

Related Concepts

►Extension Field; ►Field; ►Group

Definition

A *finite field* is a ►field with a finite number of elements.

Theory

The number of elements or the ►order of a field with a finite number of elements is always a power $q = p^k$ of a ►prime number $p \geq 2$, where $k \geq 1$. Since all finite fields with a given order are *isomorphic* (►homomorphism), it is conventional to refer to a finite field with order q as "the" finite field with that order, denoted $\mathbf{F}_q$ or $GF(q)$ (the latter meaning *Galois Field*). The prime p is the *characteristic* of the field, i.e., for all $x \in \mathbf{F}_q$, $px = 0$.

Finite fields are commonly organized into three types in cryptography:

- *Characteristic-2* or *binary fields*, where $p = 2$.
- *Prime-order fields*, where $p \geq 3$ and $k = 1$.
- *Odd-characteristic extension fields*, where $p \geq 3$ and $k > 1$.

The *multiplicative group* of a finite field, denoted $\mathbf{F}_q{}^*$, is the ►group consisting of the elements of $\mathbf{F}_q$ with multiplicative inverses, i.e., the elements except for 0, under the multiplication operation. This is a cyclic group of order $q - 1$; all the elements of the group can be obtained as powers of a single ►generator.

A classic volume in this "field" is Lidl and Niederreiter's text [1]. A book by Menezes et al. [2] gives further treatment of the cryptographic applications.

Applications

Finite fields are widely employed in cryptography. The ►IDEA and ►Rijndael/AES algorithms, for instance, both involve operations over relatively small finite fields. ►Public-key cryptography generally involves much larger finite fields. For example, the Digital Signature Algorithm (►Digital Signature Standard) operates in a ►subgroup of the multiplicative group of a prime-order finite field. ►Elliptic curve cryptography can be defined over a variety of different finite fields.

Open Problems

Efficient implementation of finite field arithmetic has been a major area of research. For a discussion, ►inversion in finite fields and rings and ►optimal extension fields.

Recommended Reading

1. Lidl R, Niederreiter H (1986) Introduction to finite fields and their applications. Cambridge University Press, Cambridge
2. Menezes AJ, Blake IF, Gao X, Mullin RC, Vanstone SA, Yaghoobian T (1992) Applications of finite fields. Kluwer, Dordrecht

FIPS 140-2

Tom Caddy
InfoGard Laboratories, San Luis Obispo, CA, USA

Synonyms

CMVP – Cryptographic module validation program; ISO 19790 2006 Security requirements for cryptographic modules

Related Concepts

►Common Criteria; ►Security Evaluation Criteria

Definition

The full name is Federal Information Processing Standard (FIPS) 140–2, titled: Security Requirements for Cryptographic Modules [1]. FIPS 140–1 and FIPS 140–2 were developed not only as documents to communicate requirements, but also as complete programs that certify products that are in full compliance with the security and assurance characteristics that are specified in the standard. The objective to provide the end customer with cryptographic products that can be trusted and used with confidence.

Background

The FIPS 140–1 and –2 program was established in response to the Information Technology Management Reform Act of 1996 and the Computer Security Act of 1987. The program is a partnership between United States National Institute of Standards and Technology and the Canadian Government Communication Security Establishment. Both the USA and Canadian organizations are integrally involved with all aspects of the program.

This document was issued May 25, 2001, and supersedes FIPS 140–1, which was published January 11, 1994 [2]. FIPS 140–1 and FIPS 140–2 were developed not only as documents to communicate requirements, but also as complete programs with the objective to provide the end customer with cryptographic products that can be trusted and used with confidence.

Theory and Applications

The full name is Federal Information Processing Standard (FIPS) 140–2, titled: Security Requirements for Cryptographic Modules [1]. The Program is called the cryptographic module validation program (CMVP). The program's intent is to balance several objectives to maximize the effectiveness for end users of the cryptographic products including security functionality, assurance, cost, and schedule. The program implementation includes testing requirements, lab accreditations, thorough report and validation result review by the regulators, and certificates of validation issued by the NIST/CSE, including an actively maintained Web site of currently approved products http://csrc.nist.gov/cryptval/.

FIPS 140–2 provides a standard that is used by government and commercial organizations when they specify that cryptographic-based security systems are to be used for providing protection of sensitive or valuable data. The protection of data, processes, and critical security parameters that is provided by a cryptographic module embedded within a security system is necessary to maintain the access control, confidentiality, and integrity of the information entrusted to that system.

FIPS 140–2 is published and maintained by the US Department of Commerce; National Institute of Standards and Technology; Information Technology Laboratory; and Cryptographic Module Validation Program. The standard is designed to be flexible enough to adapt to advancements and innovations in science and technology. NIST Policy is to have standards reviewed every 5 years in order to consider new or revised requirements that may be needed to meet technological and economic changes.

FIPS 140–2 is applicable to cryptographic-based security systems that may be used in a wide variety of computer and telecommunication applications (e.g., data storage, access control and personal identification, network communications, radio, facsimile, and video) and in various environments (e.g., centralized computer facilities, office environments, and hostile environments). The cryptographic services (e.g., encryption, authentication, digital_signature, and key_management) provided by a specific cryptographic module are selected requirements that are specific to the application. The security level (see below) to which a cryptographic module is validated is chosen to provide a level of security appropriate for the security requirements of the application in which the module is intended to be utilized and the security services that the module will provide.

It is important to note the difference between FIPS 140–2 and verification of correct cryptographic algorithm testing. Algorithm testing is limited only to verifying that a design correctly performs the logical and mathematical processes to comply with the specified algorithm functionality, such as, but not limited to:

- FIPS PUB 113, Computer Data Authentication [3]
- FIPS PUB 180–3, Secure Hash Standard SHA [4]
- FIPS PUB 186–3, Digital Signature Standard (DSS) [5]
- FIPS PUB 197, AES – Advanced Encryption Standard [6]
- FIPS PUB 198, The Keyed-Hash Message Authentication Code (HMAC) [7]

In contrast FIPS 140–2 is a comprehensive security standard that integrates module access control, key management, interface control, design assurance, operational assurance, and utilizes algorithm testing to verify that each specific algorithm used is correct.

This standard specifies the security requirements that will be satisfied by a cryptographic module. These requirements have been developed over a number of years by cryptographic and information security experts from government organizations, international representatives, the Department of Defense, users, and vendors with the objective being something that is meaningful, but also reasonable with regard to technology, usability, and affordability. The requirements have been compiled to address a broad range of threats and vulnerabilities, which trusted security focal points (modules) are subject to.

The security requirements cover areas related to the secure design and implementation of a cryptographic module. These areas include:

1. Cryptographic Module Specification
2. Cryptographic Module Ports and Interfaces
3. Roles, Services, and Authentication
4. Finite State Machine Model
5. Physical Security
6. Operational Environment
7. Cryptographic Key Management
8. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)
9. Self-Tests
10. Design Assurance
11. Mitigation of Other Attacks

The standard provides four increasing qualitative levels of security intended to cover a wide range of potential applications and environments.

Security Level 1 specifies the lowest level of security; the basic requirements are defined for a cryptographic module. No specific physical security mechanisms are required in a Security Level 1 cryptographic module beyond the basic requirement for production-grade components. Such implementations may be appropriate for low-level security applications when other controls, such as physical security, network security, and administrative procedures, are limited or nonexistent.

Security Level 2 enhances the physical security mechanisms of a Security Level 1 cryptographic module by adding the requirement for physical security in the form of tamper evidence – tamper-evident seals or pick-resistant locks are placed on covers or doors to protect against unauthorized physical access. Security Level 2 also requires the module operators to authenticate to the module to assume a specific role and perform a corresponding set of services. Level 2 allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an operating system that meets the functional requirements specified in specific Common Criteria (CC) Protection Profiles and has been evaluated to a CC assurance level 2 (EAL2 or higher).

Security Level 3 increases the physical security of the cryptographic module to inhibit the intruder from gaining access to critical security parameters (CSPs) held within the cryptographic module. Physical security mechanisms required at Security Level 3 are intended to have a high probability of detecting and responding to attempts at physical access, use or modification of the cryptographic module through the implementation of strong enclosures and tamper detection/response that includes circuitry that zeroizes all plaintext CSPs when the removable covers/doors of the cryptographic module are opened.

Level 3 requires stronger identity-based authentication mechanisms, enhancing the security provided by the role-based authentication mechanisms specified for Security Level 2. Level 3 specifies more robust key management processes including the entry or output of plaintext CSPs using split knowledge procedures or to be performed using ports that are physically separated from other ports or interfaces that are logically separated using a trusted path from other interfaces. Plaintext CSPs may be entered into or output from the cryptographic module in encrypted form (in which case they may travel through enclosing or intervening systems).

Level 3 allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an operating system that meets the functional requirements specified in specified protection profiles (PPs) if they also meet the additional functional requirement of a Trusted Path and have been evaluated at the CC EAL3 (or higher) with the additional assurance requirement of an Informal Target of Evaluation (TOE) Security Policy Model.

Security Level 4 provides the highest level of security defined in this standard. At this security level, the physical security mechanisms provide a complete envelope of protection around the cryptographic module with the intent of detecting and responding to all unauthorized attempts at physical access. Penetration of the cryptographic module enclosure from any direction has a very high probability of being detected, resulting in the immediate zeroization of all plaintext CSPs. Security Level 4 cryptographic modules are useful for operation in physically unprotected and potentially hostile environments.

Level 4 also protects a cryptographic module against a security compromise due to environmental conditions or fluctuations outside of the module's normal operating ranges for voltage and temperature.

Level 4 allows the software and firmware components of a cryptographic module to be executed on a general purpose computing system using an operating system that meets the functional requirements specified for Security Level 3 and is evaluated at the CC evaluation assurance level EAL4 (or higher).

A fundamental concept to the application of FIPS 140–2 is that of a cryptographic boundary. The cryptographic boundary shall consist of an explicitly defined perimeter that establishes the physical bounds of a cryptographic module. If a cryptographic module consists of software or firmware components, the cryptographic boundary shall contain the processor(s) and other hardware components that store and protect the software and firmware components. Hardware, software, and firmware components of a cryptographic module can be excluded from the requirements of this standard if shown that these components do not affect the security of the module. This concept allows for the evaluation of products to fixed and established confines, thereby making the process feasible. It also provides for a totally self-contained crypto module that contains enough functionality to protect itself and be able to be trusted.

For purposes of specifying physical security mechanisms, FIPS 140–2 defines three possible physical embodiments for a module and the associated security mechanisms for each:

- Single-chip cryptographic modules are physical embodiments in which a single integrated circuit is employed.

- Multiple-chip embedded cryptographic modules are physical embodiments in which two or more IC chips are interconnected and are embedded within an enclosure or a product that may not be physically protected.
- Multiple-chip standalone cryptographic modules are physical embodiments in which two or more IC chips are interconnected and the entire enclosure is physically protected.

For a module to receive a certificate of validation, it must follow a process defined by the CMVP program and be tested by an accredited laboratory. Laboratories are accredited to perform FIPS 140–2 testing based on proving to the National Voluntary Laboratory Accreditation Program (NVLAP) that they have the appropriate quality system, quality process, cryptographic skills, and FIPS 140–2 knowledge to warrant accreditation. Laboratory reaccreditation occurs on an annual basis.

Testing by the laboratory is based on a document related to FIPS 140–2 termed the Derived Test Requirements (DTR) document [8]. This document outlines the responsibilities of the test laboratory and the analysis and testing that is performed. It also describes the information and material that the vendor must provide the laboratory for the purpose of evaluating the compliance of the module to FIPS 140–2 requirements.

Acronyms

- CC – Common Criteria
- CSE – Communications Security Establishment
- CSP – Critical Security Parameters
- DSS – Digital Signature Standard
- DTR – Derived Test Requirements
- EAL – Evaluation Assurance Level
- EMC – Electromagnetic Compatibility
- EMI – Electromagnetic Interference
- ITL – Information Technology Laboratory
- FIPS – Federal Information Processing Standard
- NIST – National Institute of Standards and Technology
- NVLAP – National Voluntary Laboratory Accreditation Program
- PP – Protection Profiles
- SHA – Secure Hash Algorithm
- SPM – Security Policy Modeling
- TOE – Target of Evaluation

Recommended Reading

1. National Institute of Standards and Technology (NIST) (2001) Security requirements for cryptographic modules: federal information processing standards publication 140–142 (FIPS PUB 140–142). National Institute of Standards and Technology (NIST), Gaithersburg, http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf
2. National Institute of Standards and Technology (NIST), Computer Systems Laboratory (1994) Security requirements for cryptographic modules: federal information processing standards publication 140–1 (FIPS PUB 140–1). National Institute of Standards and Technology (NIST) and Computer Systems Laboratory, Gaithersburg, http://csrc.nist.gov/publications/fips/fips140-1/fips1401.pdf
3. National Institute of Standards and Technology (NIST) (1985) Computer data authentication: federal information processing standards publication 113 (FIPS PUB 113). http://www.itl.nist.gov/fipspubs/fip113.htm
4. National Institute of Standards and Technology (NIST) (1995) Secure hash standard (SHS): federal information processing standards publication 180–1 (FIPS PUB 180–1). National Institute of Standards and Technology (NIST), Gaithersburg, http://www.itl.nist.gov/fipspubs/fip180-1.htm
5. National Institute of Standards and Technology (NIST) (2000) Digital signature standard (DSS): federal information processing standards publication 186–2 (FIPS PUB 186–2). National Institute of Standards and Technology (NIST), Gaithersburg, http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf
6. National Institute of Standards and Technology (NIST) (1999) Announcing the Advanced Encryption Standard (AES): federal information processing standards publication 197 (FIPS PUB 197). National Institute of Standards and Technology (NIST), Gaithersburg, http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
7. National Institute of Standards and Technology (NIST) (1980) The keyed-hash message authentication code (HMAC): federal information processing standards publication 198–1 (FIPS PUB 198–1). National Institute of Standards and Technology (NIST), Gaithersburg, http://csrc.nist.gov/publications/fips/fips198-1/fips-198-1_final.htm
8. National Institute of Standards and Technology (NIST) (2003) Derived test requirements for FIPS PUB 140–2. http://csrc.nist.gov/cryptval/140-1/fips1402DTR.pdf

Firewall

NIELS PROVOS
Google INC, Mountain View, CA, USA

Definition

A *firewall* is a network device that enforces security *policy* for network traffic. The term originates from *fire wall*, a fireproof wall used as a barrier to prevent the spread of fire. An Internet firewall creates a barrier between separate networks by imposing a point of control that traffic needs to pass before it can reach a different network [2]. A firewall may limit the exposure of hosts to malicious network traffic, e.g., remote adversaries attempting to exploit security holes in vulnerable applications, by preventing certain packets from entering networks protected by the firewall.

Theory

When inspecting a network packet, a firewall decides if it should drop or forward the packet. The decision is based on a firewall's security policy and its internal state. Before forwarding a packet, a firewall may modify the packet's content. Packet inspection may occur at several different layers:

- The *link layer* provides physical addressing of devices on the same network. Firewalls operating on the link layer usually drop packets based on the *media access control* (MAC) addresses of communicating hosts.
- The *network layer* contains the *Internet protocol* (IP) headers that support addressing across networks so that hosts on different physical networks can communicate with each other.
- The *transport layer* provides data flows between hosts. On the Internet, the *transmission control protocol* (TCP) and the *user datagram protocol* (UDP) are used for this purpose. Most firewalls operate at the network and transport layer. TCP provides reliable data flow between hosts. UDP is a much simpler but unreliable transport protocol.
- The *application layer* contains application specific protocols such as the *hypertext transfer protocol* (HTTP). Inspection of application-specific protocols can be computationally expensive because more data needs to be inspected, and more states are required.

The IP header [7] contains information required by routers to forward packet between hosts. For an IPv4 header, firewalls usually inspect the following fields: header length, total length, protocol, source IP address, and destination IP address. The protocol field states which transport protocol follow after the IP header. The most frequently used transport protocols are TCP, UDP, and ICMP. While the IP protocol provides addressing between different hosts on the Internet, TCP and UDP allow the addressing of services by employing source and destination port numbers. For example, a mail client sends e-mail by contacting a mail server on destination port 25, which is used for the *simple mail transport protocol* (SMTP), and Web servers usually listen for HTTP requests on port 80. A data flow between hosts is uniquely identified by its 4-tuple:

(source IP address, source port number, destination IP address, destination port number).

A firewall's policy is described by a list of filter rules that specify which packets may pass and which packets are to be blocked. A rule contains an action and several specifiers. The specifiers describe the packets for which a rule should be applied. The action of the last matching rule determines if the firewall forwards or drops the inspected packet. Many firewalls use a format to describe filter rules that is similar to the following:

action dir interface protocol srcip srcport destip dstport flags

The first word *action* can be either *pass* or *block* and determines if a packet that matches this rule should be forwarded or dropped. These two actions are supported by all firewalls, but many modern firewall systems also support logging of packets that match specific rules, network address translation, and generating return packets. The remaining words are specifiers that restrict which packets a rule applies to. The different elements in a filter rule are explained below:

- *dir* refers to the direction a packet is taking. It may be either *incoming* or *outgoing*. Depending on the firewall solution, the direction may be specific to a network interface or to a network boundary.
- *interface* specifies the physical or virtual network device upon which the packet was received. This is often used to classify the network from which the packet arrived as internal or external.
- *protocol* allows a filer to specify the transport protocols the rule should match. Using a protocol specifier allows us to differentiate, e.g., between UDP and TCP packets.
- *srcip* specifies which source IP addresses should match the rule. This can either be a single address or IP network ranges.
- *srcport* applies only to TCP or UDP packets and determines the range of port numbers a packet should have to match the rule.
- *dstip* and *dstport* specify which destination IP addresses and destination ports to match and are very similar to the source specifiers described above.
- *flags* may contain firewall specific features. For example, the *SYN* flag indicates that rule matches only where the packet contains a TCP connection setup message (see below).

In the following example, a simple firewall configuration that blocks all traffic except SMTP connections to a central mail server is shown:

Figure 1 shows the rules of a firewall that uses only the information contained in the inspected packet to decide if the packet should be dropped or forwarded. Any incoming packet that is not sent to port 25 of the mail server matches only the first firewall rule and will be dropped. A firewall that reaches policy decisions based only on the content of the inspected packet is called *stateless*. Filtering

action	dir	interface	protocol	scrip	srcport	dstip	dstport	flags
block	in	external		all				
block	out	external		all				
pass	in	external	tcp	any	any	mail-server	25	
pass	out	external	tcp	mail-server	25	any	any	

Firewall. Fig. 1 A simple configuration that allows only traffic to and from the SMTP port of a specific mail server

action	dir	interface	protocol	scrip	srcport	dstip	dstport	flags
block	in	external		all				
block	out	external		all				
pass	in	external	tcp	any	80	any	any	
block	in	external	tcp	any	80	any	any	S/SA
pass	out	external	tcp	any	80	any	80	

Firewall. Fig. 2 A configuration that allows only outgoing HTTP traffic. The rules use TCP control flags to prevent outsiders from initiating connections to our internal network

packets that enter a network is called *ingress* filtering, and filtering packets that leave a network is called *egress* filtering.

If a policy, that allows only outgoing HTTP connections, is to be enforced, the rules for stateless firewalls are more difficult to configure. When a client initiates an HTTP connection to port 80 of a Web server, the server sends back data with source port 80. A naive firewall configuration might allow all TCP packets with source port 80 to enter the network. As a result, an adversary is allowed to initiate any kind of connection as long as the packets arrive with the correct source port. The firewall configuration can be improved by inspecting the control flags used by TCP. A packet to initiate a connection has only the *SYN* flag set. The receiver acknowledges the connection attempt by setting both the *SYN* and *ACK* flag. After a connection has been established, the *SYN* flag is not used again. A better firewall configuration is shown in Fig. 2. The third rule permits all external TCP traffic with source port 80, but the subsequent rule blocks all Web server packets that contain a SYN flag but no ACK flag. In other words, external connection attempts are denied.

A more flexible configuration is permitted by *stateful* firewalls [8] which track connection states for ICMP, TCP, and UDP packets. While UDP and ICMP are connectionless protocols, a stateful firewall may create short-lived states for them. The connection state allows a firewall to determine if a packet is part of an ongoing connection and drop packets that are outside the valid range of parameters. A stateful firewall is easier to configure because packets that match connection state are forwarded without consulting the firewall rules. The HTTP-client example from above can be realized with a stateful firewall using only one rule; see Fig. 3.

Because IP is independent of the physical network layer, it is possible that an IP packet needs to be fragmented into several smaller packets that fit within the frame size of the physical medium. The first IP fragment contains all addressing information, including ports. The firewall may enforce its policy only for the first fragment. Even if all subsequent fragments are forwarded by the firewall, without the first fragment it is not possible for the end host to completely reassemble the IP packet.

The situation is more complicated if the firewall decides to forward the first fragment. As it is possible for fragments to overlap, an adversary may decide to send an IP fragment that overlaps the transport header data of the first fragment. In that case, an adversary may be able to rewrite the port numbers and effectively bypass the firewall's security policy. Overlapping IP fragments may also confuse *network intrusion detection systems* (NIDS) [5].

There are several methods to send packets that may bypass application-level firewalls and evade intrusion detection systems [6]. Because a NIDS does not have complete knowledge of network topology and end host state, an adversary may evade security policy by exploiting ambiguities in the traffic stream monitored by the NIDS. For example, a packet may be seen by the monitoring device but never reach the end host. As a result, the traffic seen by the monitor differs from the traffic received by an end host, and the monitor may make an incorrect decision.

One possible solution to remove ambiguities from the network traffic is called *traffic normalization* [4]. It rewrites ambiguous traffic so that the interpretation of traffic at the end host is known to the NIDS. As a result, an adversary loses direct control over the specific layout of traffic, making it more difficult to evade intrusion detection

action	dir	interface	protocol	scrip	srcport	dstip	dstport	flags
block	in	external		all				
block	out	external		all				
pass	**out**	**external**	**tcp**	**any**	**any**	**any**	**80**	**keep state**

Firewall. Fig. 3 A stateful firewall is easier to configure because it keeps track of established connections. Packets that are part of an ongoing connection do not need to match any rules

systems. Recently, firewalls have started to support traffic normalization. Handley et al. enumerate methods to normalize traffic. For example, a firewall may reassemble all IP fragments into complete IP packets before forwarding them.

Firewalls may also help to limit the problem of *address spoofing* and *denial of service* attacks [3]. Address spoofing refers to IP packets that carry an incorrect IP source address. An adversary may be able to exploit trust relationships by sending packets via an external network that claim to originate from the internal network [1]. Denial of service is aimed at disrupting network services and often uses thousands of machines to generate network traffic that saturates the available bandwidth at the target host. To prevent the tracing of the attack back to the originators, denial of service attacks often use IP address spoofing.

Organizations that employ firewalls should enforce a policy that does not allow spoofed traffic to either enter or leave the network by employing both ingress and egress filtering. Denying packets with spoofed source addresses from leaving the network prevents denial of service attacks to hide their origin.

Additional Definitions

- Ingress filtering: filtering packets that enter a network.
- Egress filtering: filtering packets that leave a network.
- Stateful packet inspection: decisions to drop or forward a packet are based not only on the content of the inspected packet, but also on previous network traffic.
- Denial of service: disrupting a computer service by exhausting resources like network bandwidth, memory, or computational power.
- Network intrusion detection system: a system that monitors network traffic to identify abnormal behavior that indicates network attacks.

Recommended Reading

1. Bellovin SM (1989, April) Security problems in the TCP/IP protocol suite. ACM Comput Commun Rev 2:19/32–48
2. Cheswick WR, Bellovin SM (1994) Firewalls and Internet security repelling the Willy Hacker. Addison-Wesley, Reading, MA
3. Ferguson P, Senie D (2000) Network ingress filtering: defeating denial of service attacks which employ IP source address spoofing. RFC 2827
4. Handley M, Kreibich C, Paxson V (2001) Network intrusion detection: evasion, traffic normalization and end-to-end protocol semantics. In: Proceedings of the 10th USENIX security symposium, Aug 2001
5. Paxson V (1988) Bro: a system for detecting network intruders in real-time. In: Proceedings of the 7th USENIX security symposium, Jan 1998
6. Ptacek T, Newsham T (1998) Insertion, evasion, and denial of service: eluding network intrusion detection. Secure Networks Whitepaper, Aug 1998
7. Stevens WR (1994) TCP/IP Illustrated, vol 1. Addison-Wesley, Reading, MA
8. van Rooij G (2000) Real stateful TCP packet filtering in IP filter. In: Proceedings of the 2nd international SANE conference, May 2000

Firewall Policy Analysis

►Firewalls

Firewalls

Ehab Al-Shaer
Cyber Defense and Network Assurability (CyberDNA) Center, Department of Software and Information Systems College of Computing and Informatics; University of North Carolina Charlotte, Charlotte, NC, USA

Synonyms

Consistency verification of security policy; Firewall policy analysis; IPSec policy analysis

Definition

A network access control list or ACL, $\mathcal{P}$, is a set of ordered rules, $\mathcal{R}_i$, the defines the *transformation actions*, $\mathcal{A}_i$, (e.g., such as accept or deny in firewalls, and ESP for encryption and AH for authenticate in IPSec) that are performed on matched *traffic flows*, $\mathcal{C}_i$, defined using header information. A conflict exists in ACL policies if there are two (or more) different rules ($\mathcal{R}_i$ and $\mathcal{R}_j$) that match the same traffic flow partially or complectly (i.e., $\mathcal{C}_i \bigcap \mathcal{C}_j \neq \emptyset$). If the two rules existing in the same firewall or IPSec device, it is called *intra-policy* conflict. However, if they exist in two different devices across the network, it is called *intra-policy* conflict.

Background

Complexity of Managing Network Access Control Polices

Network security polices are essential elements in Internet security devices that provide traffic filtering, integrity, confidentiality, and authentication. Network security perimeter devices such as firewalls, IPSec, and IDS/IPS devices operate based on locally configured policies. However, configuring network security policies remains a complex and error-prone task due to the rule-dependency semantics, and the interaction between policies in the network. This complexity is likely to increase as the network size increases. A successful deployment of a network security system requires global analysis of policy configurations of all network security devices in order to avoid policy conflicts and inconsistency. Policy conflicts may cause serious security breaches and network vulnerability such as blocking legitimate traffic, permitting unwanted traffic, and insecure data transmission.

There are many challenges confronting the correctness and consistency of security policy configuration in enterprise networks. First, in a single security device, the ordering of the policy rules is critically important to determine the underlying policy semantics. An incorrect rule ordering may obsolete some critical rules and result in a security hole that can be a target for a network attack. Second, the interaction between different network security policies in a distributed network environment introduces additional challenges. For example, inconsistent rule matching between two firewalls can result in illegitimate traffic being allowed into the network, leading to serious security threats such as denial of service attacks. Also, incorrect configuration of cascaded IPSec tunnels can lead to sending confidential data as clear text violating the security policy. Third, the existence of many action types (e.g., bypass, discard, encrypt/tunnel, authenticate/transport, etc.) presents another challenge when analyzing network security policies. Policy conflicts in IPSec devices may cause either a failure in establishing secure communication, or degrade the performance due to performing unnecessary redundant security operations.

Network Access Control Background

A *security policy enforcement point* is the network element that controls the traversal of packets across network segments based on a given *network security policy*. These include firewalls and IPSec devices, which are installed either at end hosts or at intermediate network nodes. The network protection offered by firewalls/IPSec is based on requirements defined by a security policy established and maintained by a user or system administrator. In general, packets are selected for a packet transmission/protection mode based on network and transport layer header information matched against entries in the policy, i.e., transport protocol, source address and port number, and destination address and port number. Each packet is either discarded, protected and transmitted, or permitted to flow without protection. The decision is based on the policy rules that match this traffic. To define traffic protection rules, we use a generic policy format that resembles the format used in a wide range of firewall and IPSec implementations [3]. In this format, the network security policy is composed of two lists of packet-filtering rules: access list and map list.

Access list It consists of ordered filtering rules that define the actions performed on packets that satisfy the rule conditions. All traffic is matched against the access rules sequentially till a matching rule is found. The rules are ordered such that all the packets in a flow will always trigger the same rule. The rule determines the required security action, where a *protect* action means that the traffic is securely transmitted, a *bypass* action means that the traffic is transmitted unsecured, and a *discard* action means that the traffic is dropped. Such rules are used in firewall policies to permit or discard traffic, and in IPSec policies to select the traffic to be protected.

Map list The rules in this list determine the security mapping required to protect the traffic selected by the access list, also based on a set of packet filters. Each rule is given a priority and the traffic is matched against the highest priority rule first. Multiple rules can be triggered for the same flow resulting in applying more than one security transformation on the same traffic. Such rules are used in IPSec policies to specify the cryptographic transformation for a specific traffic.

Firewalls control the traversal of packets across the boundaries of a secured network based on the security policy. A firewall security policy is an access list of ordered filtering rules. Filtering actions are either *bypass*, which permits the packet to enter or leave the secure network, or *discard*, which causes the packet to be blocked and dropped. Firewalls are widely used to protect the network from external attacks like port and address scanning and denial of service attacks. They are also used to control the flow of traffic between network segments.

IPSec [6] is one of the most widely used protocols to provide secure transmission across the Internet. IPSec uses two security protocols to provide traffic security: Authentication Header (AH) and Encapsulating Security Payload (ESP). The AH protocol provides connectionless integrity, data origin authentication, and an optional anti-replay service. The ESP protocol provides confidentiality (encryption) and optional connectionless integrity, data origin authentication, and an anti-replay service. These protocols may be applied individually or in combination with each other to provide the desired security services. IPSec operations can be performed either at the traffic source and destination (transport mode) or at intermediate *security gateways* (tunnel mode), in order to allow for source-based or domain-based security. In transport mode the protocols provide protection primarily for upper layer protocols without changing the original IP headers. However, in tunnel mode, the protocols are applied to the entire IP packet and may modify the source and destination address of the original IP header to be the intermediate security gateways.

Network security policy rules can be written using the syntax shown in Fig. 1. The *access_list* is used to define firewall policies as well as IPSec protection rules, while the *map_list* is used to define IPSec transformation rules. A source or destination IP address can be specified as a single value or a range of values with a common network prefix. A source or destination port number is given as a single value or an interval of values. A *transform* is any cryptographic service that can be used to protect network traffic. These security services are practically IPSec AH and ESP either in transport or tunnel mode along with the cryptographic algorithm and the necessary cryptographic parameters. Figure 2 shows an example of

$$\begin{aligned}
access\_list &:= access\_rule[\ldots, access\_rule]\\
access\_rule &:= order, filter, action\\
filter &:= transport, src\_ip, src\_port, dst\_ip, dst\_port\\
action &:= \text{protect} \mid \text{bypass} \mid \text{discard}\\
map\_list &:= map\_rule[\ldots, map\_rule]\\
map\_rule &:= priority, filter, transform\_list\\
transform\_list &:= transform[\ldots, transform]\\
transform &:= sec\_protocol, encaps\_mode, parameters\\
sec\_protocol &:= \text{AH} \mid \text{ESP}\\
encaps\_mode &:= \text{Transport} \mid \text{Tunnel}\ tunnel\_dst
\end{aligned}$$

Firewalls. Fig. 1 The syntax of network security policy statements

a typical firewall/IPSec policy for outbound traffic. In this example, security gateways integrate firewall and IPSec functionality. The policy at each device is defined in terms of the access-list (upper section) and the map-list (lower section). In our work, we assume that inbound traffic is matched against a mirror image of the outbound IPSec policy, where the packet filters for the traffic source and destination are swapped [3]. The policy in host $A1$ requires integrity verification using AH transport for all traffic flowing to host $B3$. The policy in gateway SG_A blocks FTP traffic from flowing to any host in network B. In addition, all traffic flowing from network A to network B should be encrypted using ESP tunneling, except for the traffic coming from host $A1$.

In the rest of this chapter we proceed with the classification and analysis of conflicts in network security policies. Figure 3 shows the organization of our taxonomy of these conflicts. We divide the conflicts into two main categories: *access-list* conflicts that may exist between the rules in the access policy, and *map-list* conflicts that may exist in the map policy. We further classify the access-list conflicts into two subcategories: *intra-policy* conflicts that may exist within a single policy, and *inter-policy* conflicts between the policies in different devices. Finally, we identify the possible policy conflicts in each subcategory.

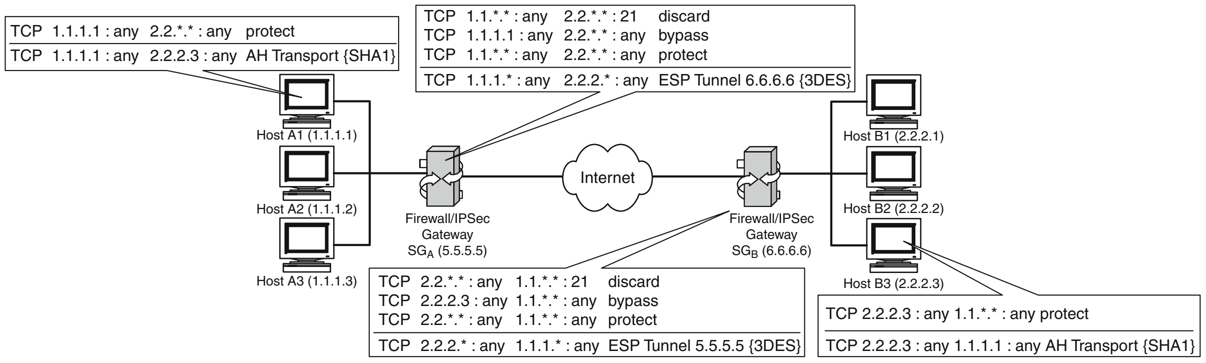
Theory and Applications

Modeling of Rule Relations

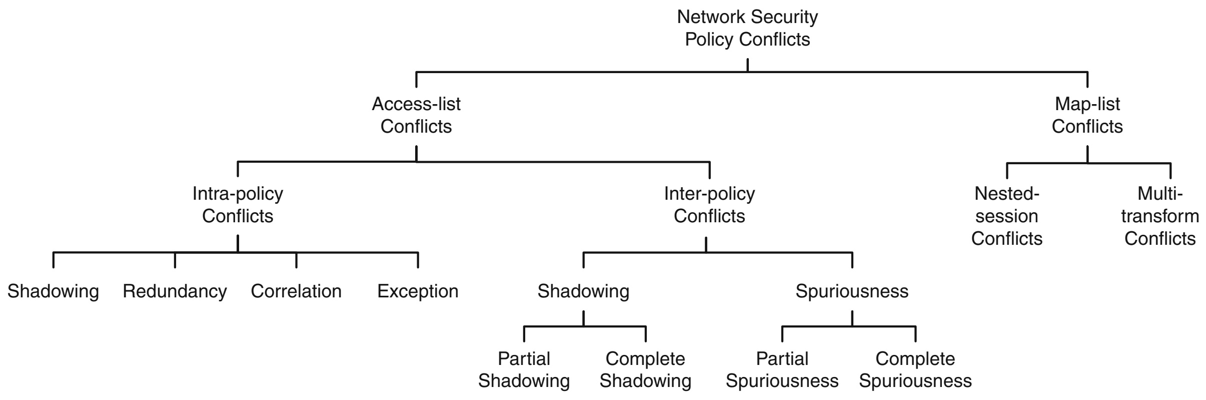
As rules are matched sequentially, the inter-rule relation or dependency is critical for determining any conflict in the security policy. In other words, if the rules are disjoint (no inter-rule relation), then any rule ordering in the security policy is valid. Therefore, classifying all types of possible relations between filtering rules is a first step to understand the source of conflicts due to policy misconfiguration. In this section we describe all the relations that exist between filtering rules by comparing the fields of filtering rules. The relation between field i in rule R_x and the corresponding field in rule R_y can be either equal, subset, superset, or distinct. For example, the IP address 140.192.61.5 is distinct from 140.192.60.5, but it is a subset of 140.192.61.*. We can then define the filtering rule relations as follows:

– *Exactly matching rules (R_x=R_y).* Rules R_x and R_y are *exactly matched* if every field in R_x is equal to the corresponding field in R_y. In the following example, Rule 1 and Rule 2 are exactly matching:

```
1: tcp 140.192.37.* any 61.20.33.* 80
2: tcp 140.192.37.* any 61.20.33.* 80
```


TCP 1.1.1.1 : any 2.2.*.* : any protect
TCP 1.1.1.1 : any 2.2.2.3 : any AH Transport {SHA1}
TCP 1.1.*.* : any 2.2.*.* : 21 discard
TCP 1.1.1.1 : any 2.2.*.* : any bypass
TCP 1.1.*.* : any 2.2.*.* : any protect
TCP 1.1.1.* : any 2.2.2.* : any ESP Tunnel 6.6.6.6 {3DES}
Host A1 (1.1.1.1)
Host A2 (1.1.1.2)
Host A3 (1.1.1.3)
Firewall/IPSec Gateway SG_A (5.5.5.5)
Internet
Firewall/IPSec Gateway SG_B (6.6.6.6)
Host B1 (2.2.2.1)
Host B2 (2.2.2.2)
Host B3 (2.2.2.3)
TCP 2.2.*.* : any 1.1.*.* : 21 discard
TCP 2.2.2.3 : any 1.1.*.* : any bypass
TCP 2.2.*.* : any 1.1.*.* : any protect
TCP 2.2.2.* : any 1.1.1.* : any ESP Tunnel 5.5.5.5 {3DES}
TCP 2.2.2.3 : any 1.1.*.* : any protect
TCP 2.2.2.3 : any 1.1.1.1 : any AH Transport {SHA1}

Firewalls. Fig. 2 Example of a typical Firewall/IPSec configuration. Only hosts *A*1 and *B*3 are IPSec capable. Security gateways SG_A and SG_B perform both firewall filtering and IPSec protection operations



Firewalls. Fig. 3 Classification chart of network security policy conflicts

– *Inclusively matching rules* ($R_x \subset R_y$). Rule R_x *inclusively matches* R_y if the rules do not exactly match and if every field in R_x is a subset or equal to the corresponding field in R_y. R_x is called the *subset match* while R_y is called the *superset match*. In the following example, Rule 3 inclusively matches Rule 4. Rule 3 is the subset match of the relation while Rule 4 is the superset match:

```
3: tcp 67.92.37.20 any 61.20.33.* 80
4: tcp 67.92.37.* any 61.20.*.* 80
```

– *Correlated rules* ($R_x \bowtie R_y$). Rules R_x and R_y are *correlated* if at least one field in R_x is a subset or partially intersects with the corresponding field in R_y, and at least one field in R_y is a superset or partially intersects with the corresponding field in R_x, and the rest of the fields are equal. This means that there is an intersection between the address space of the correlated rules although neither one is subset of the other. In the following example, Rule 5 and Rule 6 are correlated:

```
5: tcp 140.192.37.* any 61.20.*.* 0-256
6: tcp 140.192.*.* any 61.20.33.* 128-512
```

– *Disjoint rules.* Rules R_x and R_y are *completely disjoint* if every field in R_x is not a subset and not a superset and not equal to the corresponding field in R_y. However, rules R_x and R_y are *partially disjoint* if there is at least one field in R_x that is a subset or a superset or equal to the corresponding field in R_y, and there is at least one field in R_x that is not a subset and not a superset and not equal to the corresponding field in R_y. Rule 7 and Rule 8 in the following example are partially disjoint:

```
7: tcp 140.192.37.* any 61.20.*.* 80
8: tcp 140.192.37.* any 61.20.33.* 21
```

Classification of Access-List Conflicts

After identifying the possible relations between rules, we are now ready to study the types of rule conflicts. In this section, we focus the discussion on the conflicts that may exist between access list rules. These conflicts may exist between rules in the same security device (intra-policy conflicts) or in different devices (inter-policy conflicts).

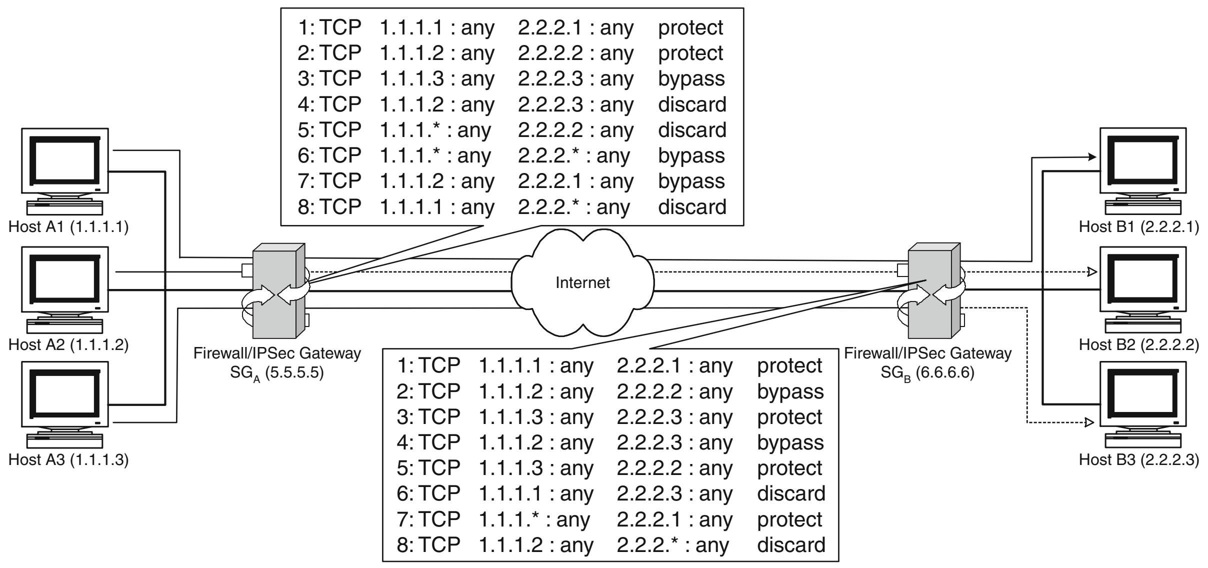
Intra-Policy Access-List Conflicts

It is very common to find filtering rules that are inter-related, i.e., exactly matched, inclusively matched, or correlated. In this case, different rule orderings may imply different and incorrect policy semantics (i.e., matching behavior). Thus, if the related rules are not carefully ordered, some rules may be concealed by other rules, resulting in an incorrect policy. In this section, we classify different conflicts that may exist among the rules in a single firewall or IPSec access policy.

Intra-policy shadowing A rule is shadowed when every packet that could match this rule is matched by some preceding rule with a different action. Consequently, the shadowed rule will never have an effect in the policy. Typically, rule R_y is shadowed by rule R_x if R_x precedes R_y and R_x *exactly* or *inclusively* matches R_y and the two rules have different actions. For example, in Fig. 4 Rule 8 in SG_A is shadowed by Rule 6 because Rule 6 will match and thereby conceal all the traffic that would match Rule 8. Shadowing is a critical error in the policy, as the shadowed rules become obsolete and never take effect. This might cause a legitimate traffic to be blocked or illegitimate traffic to be permitted. Therefore, as a general guideline, if there is an inclusive or exact match relationship between two rules, any superset (or general) rule should come after the subset (or specific) rule. It is mandatory to discover shadowed rules and alert the administrator to correct this error by reordering or removing the shadowed rule.

Intra-policy correlation As described in section "Modeling of Rule Relations", if two rules are correlated then there is some traffic that matches both rules. Thus, if the two correlated rules have different actions, then the action performed on the traffic is dependent on the ordering of the two rules. This dependency is not intuitive and leads to ambiguity in defining the security policy. A correlation conflict between two rules exists if the rules are correlated and they have different filtering actions. For example, a correlation conflict exists between Rule 7 and Rule 8 in SG_B in Fig. 4. The two rules with this ordering imply that all traffic that is coming from 1.1.1.2 and going to 2.2.2.1 is protected. However, if their order is reversed, the same traffic will be discarded. Correlation is considered a conflict warning because the correlated rules imply an action that is not explicitly handled by the policy. Therefore, in order to resolve this conflict, the correlation between rules should be discovered and the user should choose the proper rule order that complies with the security policy requirements. Otherwise, unexpected action might be performed on the traffic that matches the intersection of the correlated rules.

Intra-policy exception Exception rules are very common in network security policies. A rule is an exception of a following rule if they have different actions, and the following rule is a superset match (it could match all packets of the preceding rule). For example, in SG_A in Fig. 4, Rule 2 is an



```
1: TCP  1.1.1.1 : any   2.2.2.1 : any   protect
2: TCP  1.1.1.2 : any   2.2.2.2 : any   protect
3: TCP  1.1.1.3 : any   2.2.2.3 : any   bypass
4: TCP  1.1.1.2 : any   2.2.2.3 : any   discard
5: TCP  1.1.1.* : any   2.2.2.2 : any   discard
6: TCP  1.1.1.* : any   2.2.2.* : any   bypass
7: TCP  1.1.1.2 : any   2.2.2.1 : any   bypass
8: TCP  1.1.1.1 : any   2.2.2.* : any   discard
```

```
1: TCP  1.1.1.1 : any   2.2.2.1 : any   protect
2: TCP  1.1.1.2 : any   2.2.2.2 : any   bypass
3: TCP  1.1.1.3 : any   2.2.2.3 : any   protect
4: TCP  1.1.1.2 : any   2.2.2.3 : any   bypass
5: TCP  1.1.1.3 : any   2.2.2.2 : any   protect
6: TCP  1.1.1.1 : any   2.2.2.3 : any   discard
7: TCP  1.1.1.* : any   2.2.2.1 : any   protect
8: TCP  1.1.1.2 : any   2.2.2.* : any   discard
```

Firewalls. Fig. 4 Example IPSec access-list policy with intra- and inter-policy conflicts. *Solid* lines indicate permitted traffic, while *dotted* lines indicate blocked traffic exception of Rule 5. These two rules imply that all the traffic coming from the address 1.1.1.* to 2.2.2.2 will be discarded, except the traffic coming from 1.1.1.2. Exception is often used to exclude a specific part of the traffic from a general filtering action. This is not in general an error; however, it is important to identify exceptions because the exception rules change the policy semantics, which may or may not be desirable. This, for example, might cause accepted traffic to be blocked or denied traffic to be permitted. Thus, this type of conflict is considered a warning that is important to be highlighted to the administrator.

Intra-policy redundancy A rule is redundant when every packet that could match this rule is matched by some other rule that has a similar action. In other words, if the redundant rule is removed, the behavior of the security policy remains unchanged. Rule R_y is redundant to rule R_x if R_x precedes R_y and R_y *exactly* or *inclusively* matches R_x and the two rules have similar actions. However, rule R_x can be redundant to R_y if R_x *inclusively* matches R_y and there exists no rule in-between that is an exception or correlated with R_x. Figure 4 shows two rule redundancy examples in SG_A: Rule 3 with Rule 6 is the first case, and Rule 7 with Rule 6 is the second. Redundancy is considered a problem in firewall policies because it increases the policy size and consequently the search time and space of packet filtering, while it does not contribute to the policy semantics. Thus, it is important to discover redundant rules so that the administrator may modify or remove them altogether. In general, to avoid redundant rules, a superset rule following a subset rule should have a different filtering action.

Inter-Policy Access-List Conflicts

Even if every device policy in the network does not contain any of the intra-policy conflicts described in section "Classification of Access-List Conflicts", conflicts could exist between policies of different devices. For example, a firewall might block traffic that is permitted by another downstream firewall, or an upstream IPSec device might protect traffic that is not protected by the peer downstream device. In both cases, the traffic will not reach the destination as it will be dropped by the upstream devices. In this section, we first define criteria of inter-policy conflicts and then classify the conflicts that may exist between an upstream and a downstream access policy for both firewall and IPSec devices.

Referring to Fig. 4, we assume a traffic stream flowing from sub-domain D_A to sub-domain D_B across multiple cascaded policy enforcement points installed on the network path between the two sub-domains. At any point on this path in the direction of flow, a preceding device is called an *upstream device* whereas a following one is called a *downstream device*. Using the above network model, we can say that for any traffic flowing from sub-domain D_A to sub-domain D_B a conflict exists if one of the following conditions holds:

1. The most-downstream device permits traffic that is blocked by any of the upstream devices.
2. The most-upstream device permits traffic that is blocked by any of the downstream devices.

On the other hand, all intermediate devices should permit/bypass any traffic that is permitted/protected by the most-upstream and most-downstream device such that the flow can reach its destination.

Inter-policy shadowing This conflict is similar to the one discussed in section "Classification of Access-List Conflicts" except that it occurs between rules in two different policies/devices. Thus, inter-policy shadowing occurs if the upstream policy blocks some traffic that is permitted by the downstream policy.

There are two cases for shadowing between polices. The first case occurs if the traffic permitted (or protected) by a downstream rule, R_d, in SG_B is actually discarded by the upstream rule, R_u, in SG_A. In the second case, the traffic protection is required by R_u in SG_A but not by any R_d in SG_B. In IPSec, this causes the security association negotiation to fail and consequently results in discarding this traffic at the upstream device. In Fig. 4, Rule 4 in SG_A and Rule 4 in SG_B show an example of the first case. An example for the second case is Rule 2 in SG_A and Rule 2 in SG_B. If the rules exactly match, then *all* the traffic permitted by R_d is blocked by R_u, and we have *complete shadowing*. The two examples above are considered complete shadowing. However, if the rules inclusively match, then part of the traffic permitted by R_d is blocked by R_u, and we have *partial shadowing*. Rule 5 in SG_A and Rule 5 in SG_B are an example of partial shadowing.

Shadowing is considered a conflict because it prevents the traffic desired by some nodes from flowing to the end destination.

Inter-policy spuriousness This is another conflict that has serious consequence for network security as it allows for unwanted traffic to flow in the network. Traffic is called spurious if the upstream policy permits some traffic that is blocked by a downstream policy.

Spuriousness can occur when traffic is permitted by an upstream rule, R_u, in SG_A but discarded (or protected) by a downstream rule, R_d, in SG_B. Obviously, this traffic will not reach the destination domain D_B but yet it was unnecessarily allowed to flow through the network till it got discarded. In Fig. 4, Rule 3 in SG_A and Rule 3 in SG_B show an example of spuriousness. If the rules exactly match, then all the traffic permitted by R_u is blocked by R_d, and we have *complete spuriousness*. If the rules inclusively match, then part of the traffic permitted by R_u is blocked by R_d, and we have *partial spuriousness*. An example of partial spuriousness is Rule 6 in SG_A and Rule 6 in SG_B in the same figure.

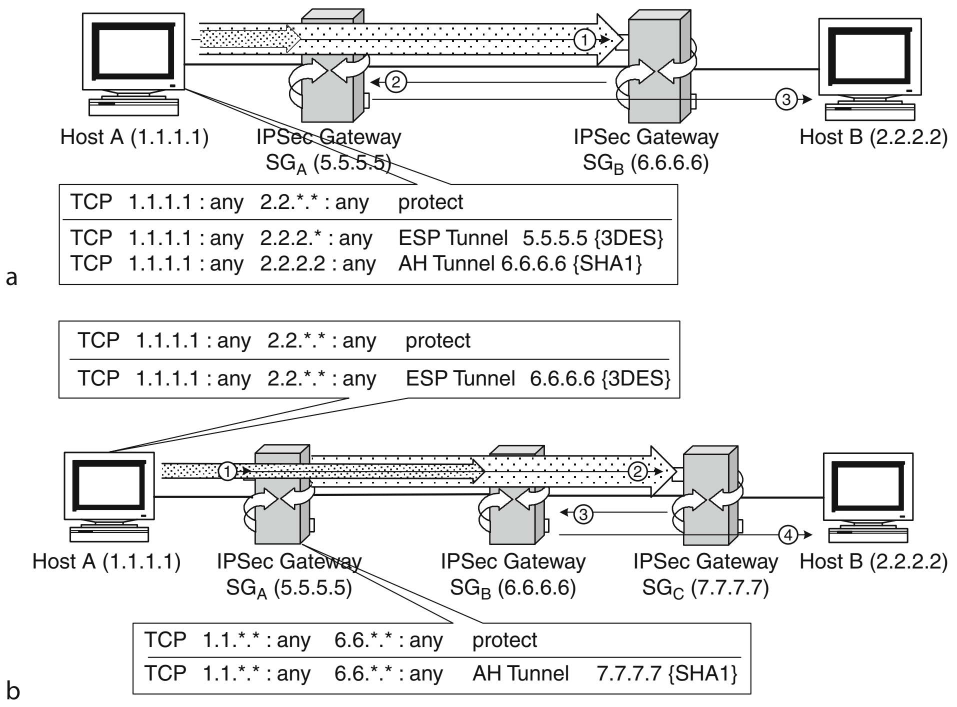
Spuriousness is a critical conflict because it allows unwanted traffic to flow across the network, increasing the network vulnerability to various network attacks such as port scanning, denial of service, etc.

Classification of Map-List Conflicts

The map-list is critical because it is the part of the policy that specifies the traffic security requirements. In this section, we identify the rule conflicts that may exist between the map list rules in a single (intra-policy) IPSec device or between multiple (inter-policy) IPSec devices. As a result of such conflicts, traffic might be transmitted insecurely or unnecessary redundant protection operations are applied to the same traffic. We also show examples of these conflicts in IPSec traffic transformation policies.

Overlapping-Session Conflicts

IPSec security policy allows for nesting multiple secure sessions or tunnels by applying a number of map-list rules to the same traffic (e.g., Fig. 5a). In case of intra-policy nested IPSec sessions, each session starts at the source (encapsulation point) and terminates at a selected peer on the path (decapsulation point) specified in the rule, and the sessions are applied in the order of their priorities in the policy. If the decapsulation point of the firstly applied session (inner tunnel) (e.g., 5.5.5.5 in Fig. 5a) happens to be located before that of a subsequent session (outer tunnel) (e.g., 6.6.6.6 in Fig. 5a) on the traffic path, then the traffic will be transmitted insecurely as clear text. This happens simply because the destination peer of the outer tunnel in this case returns the traffic after decapsulation back to the destination peer of the inner tunnel, which in turn decapsulates and forwards the traffic to the destination as clear text. This tunnel overlapping conflict occurs because the rules were not ordered correctly in the map-list such that the priorities of IPSec sessions terminating at further points from the source are higher than the priorities of the ones with closer termination points. The example in Fig. 5a shows two map rules applying to the traffic flowing from host A to B. The first rule encapsulates the traffic in a tunnel


Host A (1.1.1.1)
IPSec Gateway SG_A (5.5.5.5)
IPSec Gateway SG_B (6.6.6.6)
Host B (2.2.2.2)
a
Host A (1.1.1.1)
IPSec Gateway SG_A (5.5.5.5)
IPSec Gateway SG_B (6.6.6.6)
IPSec Gateway SG_C (7.7.7.7)
Host B (2.2.2.2)
b

a

TCP	1.1.1.1 : any	2.2.*.* : any	protect
TCP	1.1.1.1 : any	2.2.2.* : any	ESP Tunnel 5.5.5.5 {3DES}
TCP	1.1.1.1 : any	2.2.2.2 : any	AH Tunnel 6.6.6.6 {SHA1}

b

TCP	1.1.1.1 : any	2.2.*.* : any	protect
TCP	1.1.1.1 : any	2.2.*.* : any	ESP Tunnel 6.6.6.6 {3DES}

TCP	1.1.*.* : any	6.6.*.* : any	protect
TCP	1.1.*.* : any	6.6.*.* : any	AH Tunnel 7.7.7.7 {SHA1}

Firewalls. Fig. 5 Examples for overlapping-session conflicts in the policy of (**a**) single device, and (**b**) multiple devices. The block arrows indicate the tunnel between two end-points. The line arrows indicate the traffic flow path with the sequence of events circled to SG_A, then the second rule re-encapsulates the traffic in another tunnel to SG_B. However, this rule ordering results in incorrect nesting that causes the traffic to be sent back from SG_B to SG_A, and then decapsulated and forwarded as clear text to B.

Similarly, the same problem might happen due to inter-policy conflict between multiple IPSec gateways. However, in this case the only difference is the encapsulation point (start of the tunnel) might be at different point on the path. Figure 5b shows an example of two overlapping tunnels: from host A to 6.6.6.6 and from 5.5.5.5 to 7.7.7.7. As in the previous case, the outer tunnel decapsulation peer (7.7.7.7) comes after the inner tunnel decapsulation peer (6.6.6.6) on the path, causing a similar security violation. The example in Fig. 5b illustrates this case. In this example, two IPSec sessions are used to protect the traffic flowing from A to B. The first session starts at A and encapsulates the traffic in a tunnel terminating at SG_B. The second session starts at SG_A and re-encapsulates the traffic in another tunnel terminating at SG_C. The traffic is first received and decapsulated by SG_C and then forwarded back to SG_B. SG_B decapsulated the traffic and forwards it to B as clear text.

In order to assure correct overlapping of inter-policy tunnels, we must guarantee the decapsulation peers of the outer tunnels come first in the path. In other words, the packets should be decapsulated in reverse order of their encapsulation at subsequent points on the traffic path. This requires that the sessions for the same traffic must be either completely nested (i.e., the end-points of one session lie between the end-points of the other session) or cascaded (i.e., none of the end-points of any session lies between the end-points of the other session). Otherwise, the traffic will be sent unsecured over parts of the path due to this conflict.

So in general, by looking at any IPSec policy, this conflict exists if two rules match a common flow, and the tunnel end-point of the firstly applied rule comes before the tunnel end-point of the following rule in the path from source to destination. Notice that this conflict can only occur with two tunneled transforms or with a transport transform followed by a tunnel.

Multi-Transform Conflicts

IPSec also allows for multiple transforms to be applied by the same gateway or multiple gateways in the path on the same flow from source to destination. This gives the user the flexibility to combine different IPSec protection actions on the same traffic. However, some of these combinations provide weak protection, such as applying ESP transport after AH transport because ESP transport does not provide IP header protection. Moreover, other combinations may cause extra overhead without any further improvement in the traffic protection, particularly if the new protection is weaker than the existing one. For example, applying an AH tunnel on traffic already encapsulated in an ESP tunnel does not improve the security protection.

The multi-transform conflict occurs when two rules match a common flow, and the secondly applied rule uses a weaker transform on top of a stronger one applied by the other rule. For flexibility, the strength of any transform can be user-defined such that if a transformation has a larger strength value, then it provides better protection, and vice versa.

Open Problems

Future firewall policies will allow multiple actions that are not necessary mutually exclusive. In this case, actions are not necessary contradiction, thereby new conflict analysis need to be investigated to consider this kind of polices.

Experimental Results

In this section, we evaluate the effectiveness of our classification in discovering the conflicts in manually produced security policies. We implemented a set of conflict discovery algorithms to discover the security policy conflicts studied in sections "Classification of Access-List Conflicts" and "Classification of Map-List Conflicts" [5]. In these algorithms, we represent the security policy as Boolean expressions in order to to enable the analysis of policy semantics and the identification of rule conflicts using Boolean operators such as assignment and implication. We use Ordered Binary Decision Diagram (OBDD) [5] to represent and manipulate the policy expressions. We implemented these algorithms in a software tool called the "Security Policy Advisor" or SPA. In this section, we present our evaluation of the usability of the policy analysis techniques described in this paper.

To assess the practical value of our techniques, we used the SPA tool to analyze real firewall and IPSec policy rules in our university network as well as in some local industrial networks in the area. In many cases, the SPA has proven to be effective by discovering many policy conflicts that were not discovered through human visual inspection. We made an attempt to quantitatively evaluate the practical usability of the SPA by conducting a set of experiments that consider the level of network administrator expertise and the frequency of occurrence of each conflict type. In this experiment, we created two IPSec policy exercises and recruited 38 network administrators with varying level of expertise in the field (7 experts, 12 intermediate

Firewalls. Table 1 The percentage of administrators who created intra- and inter-policy IPSec conflicts

	Intra-Policy Conflicts			Inter-Policy Conflicts		
Experience	Access-list Conflicts	Overlapping-session Conflicts	Multi-transform Conflicts	Access-list Conflicts	Overlapping-session Conflicts	Multi-transform Conflicts
Expert	14%	14%	0%	29%	14%	14%
Intermediate	42%	33%	8%	50%	33%	17%
Beginner	84%	63%	16%	90%	53%	16%
Conflict type %	19%	9%	7%	38%	16%	11%

and 19 beginners) to complete each exercise. The exercise included writing IPSec access list and map list rules based on a set of security policy requirements for nine interconnected networks with 12 IPSec security gateways (intermediate and end-point gateways.) We then used the SPA tool to analyze the rules and count different types of conflicts. The experimental results in Table 1 show the percentage of individuals who introduced various types of conflicts in their IPSec policy configurations.

These results show clearly that even the expert administrators created policy conflicts. A total of about 29% of experts created intra-policy and inter-policy conflicts. This figure is even much higher for intermediate and beginner administrators (a total of about 67% and 90% created intra-policy conflicts, and 75% and 95% created inter-policy conflicts, respectively). An interesting observation is that most of the persons made configuration mistakes in configuring access-list and overlapping-session rules. The table also shows the average ratio of each conflict type relative to the total number of conflicts discovered for each class of administrators. The results clearly indicate that access-list conflicts dominate the misconfiguration errors made by administrators (19% intra-policy and 38% inter-policy access-list conflicts).

Recommended Reading

1. Al-Shaer ES, Hamed HH (2004) Discovery of policy anomalies in distributed firewalls. In: Proceedings of IEEE INFOCOM'04, Hong Kong, Mar 2004, pp 2605–2626
2. Al-Shaer E, Marrero W, El-Atway A, AlBadawi K (2009) Network configuration in a box: towards end-to-end verification of network reachability and security. In: Proceedings of 17th international conference on network communications and protocol (ICNP'09), Princeton, pp 123–132
3. Configuring IPSec Network Security. Cisco IOS security configuration guide, Release 12.2, Cisco Systems, San Jose
4. Gouda MG, Liu AX, Jafry M (2008) Verification of distributed firewalls. In: Proceedings of IEEE GLOBECOM, New Orleans, Nov 2008, pp 1–5
5. Hamed H, Al-Shaer E, Marrero W (2005) Modeling and verification of IPSec and VPN security policies. In: Proceedings of international conference on netwrok communications and protocol (ICNP'05), Boston
6. Kent S, Atkinson R (1998) Security architecture for the internet protocol. RFC-2401, IETF, Nov 1998
7. Narain S, Levin G, Malik S, Kaul V (2008) Declarative infrastructure configuration synthesis and debugging. J Netw Syst Manag 16:235–258
8. Yuan L, Mai J, Su Z, Chen H, Chuah C, Mohapatra P (2006) FIREMAN: A frameworkkit for firewall modeling and analysis. In: 27th IEEE symposium on security and privacy, Oakland, May 2006
9. Zhang B, Al-Shaer ES, Jagadeesan R, Riely J, Pitcher C (2007) Specifications of a high-level conflict-free firewall policy language for multidomain networks. In: 12th ACM symposium on access control models and technologies (SACMAT 2007), France, June 2007, pp 185–194

Fixed Window Exponentiation

►2^k-Ary Exponentiation

Fixed-Base Exponentiation

André Weimerskirch
escrypt Inc., Ann Arbor, MI, USA

Related Concepts

►Binary Exponentiation; ►Diffie–Hellman Key Agreement; ►Fixed-Exponent Exponentiation; ►k-ary Exponentiation; ►Windows Exponentiation

Definition

The exponentiation of a fixed base element $g \in G$, with G some group, by an arbitrary positive integer exponent e.

Theory

There are many situations where an exponentiation of a fixed base element $g \in G$, with G some group, by an arbitrary positive integer exponent e is performed. Fixed-base exponentiation aims to decrease the number of multiplications compared to general exponentiation algorithms such as the ►binary exponentiation algorithm. With a fixed base, precomputation can be done once and then used for many exponentiations. Thus the time for the precomputation phase is virtually irrelevant. Using precomputations with a fixed base was first introduced by Brickell, Gordon, McCurley, and Wilson (and thus it is also referred to as BGMW method) [1]. In the basic version, values $g_0 = g$, $g_1 = g^2, g_2 = g^{2^2}, \ldots, g_t = g^{2^t}$ are precomputed, and then the binary exponentiation algorithm is used without performing any squarings. Having an exponent e of bit-length $n+1$, such a strategy requires on average $n/2$ multiplications whereas the standard binary exponentiation algorithm requires $3/2\ n$ multiplications. However, there is quite some storage required for all precomputed values, namely, storage for $t+1$ values. The problem of finding an efficient algorithm for fixed-base exponentiation can be rephrased as finding a short vector-addition chain for given base elements $g_0, g_1, \ldots, g_t$ (cf. fixed-exponent exponentiation). Note that there is always a trade-off between the execution time of an exponentiation and the number t of precomputed group elements.

In order to reduce the computational complexity, one might use a precomputed version of the k-ary exponentiation by making the precomputation phase only once. However, there is time saved by multiplying together powers with identical coefficients, and then raising the intermediate products to powers step by step. The main idea of the *fixed-base windowing method* is that $g^e = \prod_{i=0}^{t} g_i^{e_i} = \prod_{j=1}^{h-1}(\prod_{e_i=j} g_i)^j$ where $0 \le e_i < h$ [1]. Assume that g^e is to be computed where g is fixed. Furthermore, there is a set of integers $\{b_0, b_1, \ldots, b_t\}$ such that any appropriate positive exponent e can be written as $e = \sum_{i=0}^{t} e_i b_i$, where $0 \le e_i < h$ for some fixed positive integer h. For instance, when choosing $b_i = 2^i$ this is equivalent to the basic BGMW method described above. Algorithm 1 takes as input the set of precomputed values $g_i = g^{b_i}$ for $0 \le i \le t$, as well as h and e.

Algorithm 1 Fixed-base windowing method

```
INPUT: a set of precomputed values $\{g^{b_0}, g^{b_1}, \ldots, g^{b_t}\}$, the exponent $e = \sum_{i=0}^{t} e_i b_i$, and the positive integer $h$
OUTPUT: $g^e$
1. $A \leftarrow 1, B \leftarrow 1$
2. For $j$ from $(h-1)$ down to 1 do
   2.1 For each $i$ for which $e_i = j$ do $B \leftarrow B \cdot g^{b_i}$
   2.2 $A \leftarrow A \cdot B$
3. Return $A$
```

This algorithm requires at most $t+h-2$ multiplications, and there is storage required for $t+1$ precomputed group elements. The most common version of this method is the case where the exponent e is represented in radix b, where b is a power of 2, i.e., $b = 2^w$. The parameter w is also called the window size. The exponent is written as $e = \sum_{i=0}^{t} e_i (2^w)^i$ or $(e_t \ldots e_1 e_0)_{2^w}$ where $t+1 = \lceil n/w \rceil$ and $b_i = (2^w)^i$ for $0 \le i \le t$, and $h = 2^w$. Then on average there are $(t+1)(2^w-1)/2^w + 2^w - 3$ multiplications required. Consider the following example [5] for $e = 862$ and $w = 2$, i.e., $b = 4$. Then $b_i = 4^i, 0 \le i \le 4$, such that the values g^1, g^4, g^{16}, g^{64}, and g^{256} are precomputed. Furthermore it is $t = 4$ and $h = 4$. The following Table 1 displays the values of A and B at the end of each iteration of Step 2:

Fixed-Base Exponentiation. Table 1 Example for the windowing method

j	-	3	2	1
B	1	$g^4 g^{256} = g^{260}$	$g^{260} g = g^{261}$	$g^{261} g^{16} g^{64} = g^{341}$
A	1	g^{260}	$g^{260} g^{261} = g^{521}$	$g^{521} g^{341} = g^{862}$

A method to reduce the required memory storage for precomputation even further was proposed by Lim and Lee [6] which is called *fixed-base comb method*. Here, the binary representation of the exponent e is written in h rows such that there is a matrix EA (exponent array) established. Then v columns of the matrix are processed one at a time. Assume that the exponent is written as $e = (e_n \ldots e_1 e_0)_2$. Then select an integer h (the number of rows of EA) with $1 \le h \le n+1$ and compute $a = \lceil (n+1)/h \rceil$ (the number of columns of EA). Furthermore, select an integer v (the number of columns of EA that are processed at once) with $1 \le v \le a$, and compute $b = \lceil a/v \rceil$ (the number of processing steps). Let $X = (R_{h-1} || R_{h-2} || \ldots || R_0)$ be a bit-string formed from e by padding e on the left with 0's such that X has bit-length ah and such that each R_i is a bit-string of length a. Form the $h \times a$ array EA where row i of EA is the bit-string R_i. The fixed-base comb method algorithm has two phases. First, there is a precomputation phase that is done only once for a fixed base element g, and then there is the exponentiation phase that is done for each exponentiation. Algorithm 2 [5] describes the precomputation phase.

Algorithm 2 Fixed-base comb method - precomputation phase

```
INPUT: a group element $g$ and parameters $h, v, a$, and $b$
OUTPUT: $\{G[j][i] : 1 \le i < 2^h, 0 \le j < v\}$
1. For $i$ from 0 to $(h-1)$ do
   1.1 $g_i \leftarrow g^{2^{ia}}$
2. For $i$ from 1 to $(2^h - 1)$ (where $i = (i_{h-1} \ldots i_0)_2$), do
   2.1 $G[0][i] \leftarrow \prod_{j=0}^{h-1} g_j^{i_j}$
   2.2 For $j$ from 1 to $(v-1)$ do
      2.2.1 $G[j][i] \leftarrow (G[0][i])^{2^{jb}}$
3. Return $G[j][i]$ for $1 \le i < 2^h, 0 \le j < v$
```

Now let $I_{j,k}, 0 \leq k < b, 0 \leq j < v$ be the integer whose binary representation is column $(jb + k)$ of EA, where column 0 is on the right and the least significant bit of a column is at the top. Algorithm 3 displays the fixed-base comb method exponentiation phase.

Algorithm 3 Fixed-base comb method - exponentiation phase

```
INPUT: a group element $g$ and an exponent $e$ as well as the precomputed values $G[i][j]$
OUTPUT: $g^e$
1.  $A \leftarrow 1$
2.  For $k$ from $(b-1)$ down to 0 do
    2.1  $A \leftarrow A \cdot A$
    2.2  For $j$ from $(v-1)$ down to 0 do
         2.2.1  $A \leftarrow G[j][I_{j,k}] \cdot A$
3.  Return $A$
```

The number of multiplications required in the computation phase is at most $a + b - 2$ of which there are at least $b-1$ squarings. Furthermore, there is storage required for $v(2^h - 1)$ precomputed group elements. Note that the required computational complexity depends on h and v, i.e., on the available memory capacity for storing precomputed elements. In practice, values $h = 4$ or 8 and $v = 1$ or 2 offer a good trade-off between running time and memory requirements. Again, assume $e = 862 = (1101011110)_2$, i.e., $t = 9$. Choose $h = 3$ and thus $a = 4$, and choose $v = 2$ such that $b = 2$. In the first phase Algorithm 2 is applied. Table 2 displays the precomputed values. Here, all possible values

Fixed-Base Exponentiation. Table 2 Example for fixed-base comb method precomputation

i	1	2	3	4	5	6	7
$G[0][i]$	g_0	g_1	g_1g_0	g_2	g_2g_0	g_2g_1	$g_2g_1g_0$
$G[1][i]$	g_0^4	g_1^4	$g_1^4g_0^4$	g_2^4	$g_2^4g_0^4$	$g_2^4g_1^4$	$g_2^4g_1^4g_0^4$

Fixed-Base Exponentiation. Table 3 Example for exponent array *EA*

$a = 4$

$I_{1,1}$	$I_{1,0}$	$I_{0,1}$	$I_{0,0}$
$e_3 = 1$	$e_2 = 1$	$e_1 = 1$	$e_0 = 0$
$e_7 = 0$	$e_6 = 1$	$e_5 = 0$	$e_4 = 1$
0	0	$e_9 = 1$	$e_8 = 1$

$h = 3$

$v = 2$

$b = \lceil a/v \rceil = 2$

Fixed-Base Exponentiation. Table 4 Example for fixed-base comb method exponentiation

$A = g_0^{I_0} g_1^{I_1} g_2^{I_2}$				
k	j	I_0	I_1	I_2
1	-	0	0	0
1	1	4	0	0
1	0	5	0	1
0	-	10	0	2
0	1	14	4	2
0	0	14	5	3

that might occur in a column of the EA matrix are precomputed. Note that the values of the second row are the values of the first one to the power of $a = 4$ such that later on two columns can be processed at a time. Recall that $g_i = g^{2^{ia}}$.

Now form the bit-string $X = (001101011110)_2$ with two padded zeros. Table 3 displays the exponent array EA. Note that the least significant bit of e is displayed in the upper right corner of EA and the most significant bit in the lower left corner.

Finally, Algorithm 3 is performed. Table 4 displays the steps of each iteration. Note that only the powers of the three base values g_i are displayed.

The last row of the table corresponds to $g^e = g_0^{I_0} g_1^{I_1} g_2^{I_2} = g^{14} g^{16 \cdot 5} g^{256 \cdot 3} = g^{862}$. The fixed-base comb method is often used for implementations as it promises the shortest running times at given memory constraints for the precomputed values. A compact description of the algorithm can be found in [4].

Further examples and explanations can be found in [3, 5]. An improvement of the fixed-base windowing method, which is called *fixed-base Euclidean method*, was proposed by de Rooij [2]. However, in most situations the fixed-base comb method performs more efficiently.

Applications

A popular application of fixed-base exponentiation is in elliptic curve cryptography, for instance for Diffie-Hellman key agreement and ECDSA signature verification.

Recommended Reading

1. Brickell EF, Gordon DM, McCurley KS, Wilson DB (1992) Fast exponentiation with precomputations. In: Proceedings of Eurocrypt '92, LNCS 658, Springer-Verlag, Berlin, Heidelberg
2. de Rooij P (1994) Efficient exponentiation using precomputation and vector addition chains. In: Proceedings of Eurocrypt '94, LNCS 950, Springer-Verlag, Berlin, Heidelberg
3. Gordon DM (1998) A survey of fast exponentiation methods. J Algorithms 27:129–146
4. Hankerson D, Hernandez JL, Menezes A (2000) Software implementation of elliptic curve cryptography over binary fields. In:

Proceedings of cryptographic hardware and embedded systems, CHES 2000, LNCS 1965, Springer-Verlag, London, UK

5. Menezes AJ, van Oorschot PC, Vanstone SA (1996) Handbook of applied cryptography. CRC Press, Boca Raton, FL
6. Lim C, Lee P (1994) More flexible exponentiation with precomputation. In: Proceedings of Crypto '94, LNCS 839, Springer-Verlag, London, UK

Fixed-Exponent Exponentiation

André Weimerskirch
escrypt Inc., Ann Arbor, MI, USA

Related Concepts

►Binary Exponentiation; ►ElGamal Decryption; ►RSA

Definition

The exponentiation of a random element $g \in G$, with G some group, by a fixed integer exponent e.

Theory

There are several situations where an exponentiation g^e of an arbitrary element $g \in G$, with G some group, by a fixed exponent e needs to be performed. Fixed-exponent exponentiation algorithms aim to decrease the number of multiplications compared to general exponentiation algorithms such as the binary exponentiation algorithm. They are based on the fact that certain precomputations can be performed once for a fixed exponent. The problem of finding the smallest number of multiplications to compute g^e is equivalent to finding the shortest *addition chain* of e. An addition chain is a sequence of numbers such that each number is the sum of two previous ones and such that the sequence starts with 1. For instance, an addition chain of 19 is $V = (1, 2, 4, 5, 9, 10, 19)$.

Definition 1 *An* addition chain *V of length s for a positive integer e is a sequence $u_0, u_1, \ldots, u_s$ of positive integers, and an associated sequence $w_1, \ldots, w_s$ of pairs $w_i = (i_1, i_2)$, $0 \le i_1, i_2 < i$, having the following properties: (i) $u_0 = 1$ and $u_l = e$; and (ii) for each $u_i, 1 \le i \le s, u_i = u_{i_1} + u_{i_2}$.*

Fixed exponents are considered here, and the time for precomputations is not taken into account here. However, determining the shortest addition chain for e is believed to be computationally infeasible in most cases for chains of relevant length. Thus in practice there are heuristics used to obtain nearly optimal addition chains. Knuth [4] describes several such heuristics. It is wise to implement multiple heuristics in order to compare the results, and finally to choose the shortest addition chain. Such precomputation can be computationally very demanding, though. After an addition chain for an exponent e had been obtained, exponentiation can be performed as described in Algorithm 1.

Algorithm 1 Addition Chain Exponentiation

```
INPUT: a group element $g$, an addition chain $V = (u_0, u_1, \ldots, u_s)$ of length $s$ for a positive integer $e$, and the associated sequence $(w_1, \ldots, w_s)$, where $w_i = (i_1, i_2)$
OUTPUT: $g^e$
1. $g_0 \leftarrow g$
2. For $i$ from 1 to $s$ do
  2.1 $g_i \leftarrow g_{i_1} \cdot g_{i_2}$
3. Return $(g_s)$
```

Algorithm 1 computes g^e for a precomputed addition chain for e of length s with s multiplications. For instance, for above example of $e = 19$ it is $V = (1, 2, 4, 5, 9, 10, 19)$. Algorithm 1 then works as follows:

i	0	1	2	3	4	5	6
w_i	-	$(0,0)$	$(1,1)$	$(0,2)$	$(2,3)$	$(3,3)$	$(4,5)$
g_i	g	g^2	g^4	g^5	g^9	g^{10}	g^{19}

In some cases *addition-subtraction chains* might be used to shorten the length of a chain. In such cases a number of the chain is the sum or subtraction of two previous elements in the chain. For instance, the shortest addition chain for 31 is $V = (1, 2, 3, 5, 10, 11, 21, 31)$ whereas there exists a shorter addition-subtraction chain $C' = (1, 2, 4, 8, 16, 32, 31)$ [3]. However, addition-subtraction chains only make sense in cases where an inversion in the underlying group is computationally cheap. Thus, addition-subtraction chains are not used for exponentiation in an RSA modulus but might be applied to elliptic curve operations.

There are two generalized versions of addition chains. An *addition sequence* is an addition chain $V = (u_0, \ldots, u_s)$ such that it contains a specified set of values $r_1, \ldots, r_t$. They are used when an element g needs to be raised to multiple powers $r_i, 1 \le i \le t$. Especially when the exponents $r_1, r_2, \ldots, r_t$ are far apart, an addition sequence might be faster in such a case. Finding the shortest addition sequence is known to be NP-complete [2] and thus heuristics are used to find short sequences.

In cases of simultaneous exponentiations such as in the ►digital signature standard (DSS), a generalized addition chain called *vector-addition chain* can be applied. These are used to compute $g_0^{e_0} g_1^{e_1} \ldots g_{k-1}^{e_{k-1}}$ where the g_i's are arbitrary elements in G and the e_i's are fixed positive integers. In a vector addition chain, each vector is the sum of two previous ones. For instance, a vector-addition chain C of the vector $[15, 5, 12]$ is $([1,0,0],[0,1,0],[0,0,1],[1,0,1],[2,0,2],[2,1,2],[3,1,2],[5,2,4],[6,2,5],[12,4,10],[15,5,12])$.

Definition 2 *Let s and k be positive integers and let v_i denote a k-dimensional vector of non-negative integers. An ordered set $V = \{v_i : -k+1 \le i \le s\}$ is called a* vector-addition chain *of length s and dimension k if V satisfies the following: (1) Each $v_i, -k+1 \le i \le 0$, has a 0 in each coordinate position, except for coordinate position $i+k-1$, which is a 1. (Coordinate positions are labelled 0 through $k-1$.); and (2) For each $v_i, 1 \le i \le s$, there exists an associated pair of integers $w_i = (i_1, i_2)$ such that $-k+1 \le i_1, i_2 < i$ and $v_i = v_{i_1} + v_{i_2}$ ($i_1 = i_2$ is allowed).*

Again, the time for precomputations is irrelevant. There is a one-to-one correspondence between vector-addition chains and addition sequences [6]. Hence determining the shortest vector-addition chain is NP-complete and thus heuristics are used to obtain short vector-addition chains [1]. Having a vector-addition chain, exponentiation can be performed as shown in Algorithm 2. The algorithm needs s multiplications for a vector-addition chain of length s to compute $g_0^{e_0} g_1^{e_1} \ldots g_{k-1}^{e_{k-1}}$ for arbitrary base and fixed exponents.

Algorithm 2 Vector-Addition Chain Exponentiation

```
INPUT: group elements $g_0, g_1, \ldots, g_{k-1}$ and a vector-addition chain $V$ of length $s$ and dimension $k$ with associated sequence $w_1, \ldots, w_s$, where $w_i = (i_1, i_2)$.
OUTPUT: $g_0^{e_0} g_1^{e_1} \ldots g_{k-1}^{e_{k-1}}$ where $v_s = (e_0, \ldots, e_{k-1})$.
1. For $i$ from $(-k+1)$ to 0 do
  1.1 $a_i \leftarrow g_{i+k-1}$
2. For $i$ from 1 to $s$ do
  2.1 $a_i \leftarrow a_{i_1} \cdot a_{i_2}$
3. Return $(a_s)$
```

An overview of addition chains is given by Knuth [4]. Further examples of fixed-exponent exponentiation can be found in [3, 5]. A lower bound for the shortest length of addition chains was proven by Schönhage [7], an upper bound is obtained by constructing an addition chain of e from its binary representation, i.e., by the binary exponentiation algorithm. Yao proved bounds for addition sequences [8].

Applications

Fixed-exponent exponentiation can be used in RSA encryption using the fixed public exponent, and in RSA decryption using the fixed secret exponent. The mechanism can also be used in ElGamal decryption and elliptic curve operations. Simultaneous exponentiation with more than one fixed exponent can be applied to the digital signature standard (DSS).

Recommended Reading

1. Bos J, Coster M (1990) Addition chain heuristic. In: Proceedings of Crypto '89, LNCS 435, Springer-Verlag, Heidelberg
2. Downey P, Leong B, Sethi R (1981) Computing sequences with addition chains. SIAM J Comput 10:638–646
3. Gordon DM (1998) A survey of fast exponentiation methods. J Algorithms 27:129–146
4. Knuth DE (1997) The art of computer programming, vol 2: Seminumerical algorithms, 3rd edn. Addison-Wesley, Boston, MA
5. Menezes AJ, van Oorschot PC, Vanstone SA (1996) Handbook of applied cryptography. CRC Press, Boca Raton, FL
6. Olivos J (1981) On vectorial addition chains. In J Algorithms 2:13–21
7. Schönhage A (1975) A lower bound for the length of addition chains. Theor Comput Science 1:1–12
8. Yao AC (1976) On the evaluation of powers. SIAM J Comput 5:100–103

Flexible Authorization Framework (FAF)

Pierangela Samarati
Dipartimento di Tecnologie dell'Informazione (DTI), Università degli Studi di Milano, Crema (CR), Italy

Related Concepts

►Authorizations; ►Discretionary Access Control Policies (DAC); ►Logic-Based Authorization Languages

Definition

Flexible Authorization Framework (FAF) is a powerful and elegant logic-based framework where authorizations are specified in terms of a locally stratified rule-based logic. FAF is based on an access control model that does not depend on any policy but is capable of representing any policy through the syntax of the model.

Background

The definition of access control policies (and their corresponding models) is far from being a trivial process. One of the major difficulty lies in the interpretation of, often complex and sometimes ambiguous, real-world security policies and in their translation in well-defined and unambiguous rules enforceable by a computer system. Many real-world situations have complex policies, where access decisions depend on the application of different rules coming, for example, from laws, practices, and organizational regulations. A security policy must capture all the different regulations to be enforced and, in addition, must also consider possible additional threats due to the use of a computer system. Given the complexity of the scenario, there is a need for flexible, powerful, and expressive access control services to accommodate all the different requirements that may need to be expressed, while at the same time be simple both in terms of use (so that specifications can be kept under control) and implementation (so to allow for its verification). Logic languages are particularly attractive as policy specification languages. One main advantage is that the semantic of a logic language is clear and unambiguous. Also, logic languages are very expressive, can be used to represent any kind of policy, and are declarative, thus offering a better abstraction level than imperative programming languages.

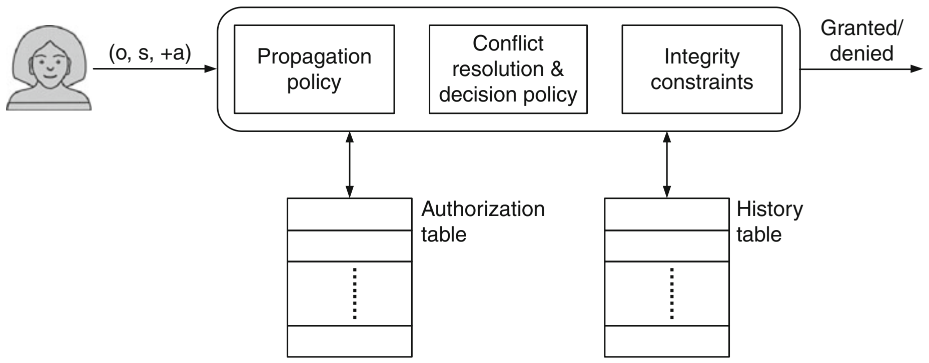
Although several logic-based authorization models and access control mechanisms have been implemented, which allow the expression of different kinds of authorization implications, constraints on authorizations, and access control policies, the authorization specifications may result difficult to understand and manage. Also, the trade-off between expressiveness and efficiency seems to be strongly unbalanced: the lack of restrictions on the language results in the specification of models that may not even be decidable and implementable in practice.

Starting from these observations, in [1] the authors propose a logic-based language that attempts to balance flexibility and expressiveness on the one side, and easy management and performance on the other. The language allows the representation of different policies and protection requirements, while at the same time providing understandable specifications, clear semantics (guaranteeing therefore the behavior of the specifications), and bearable data complexity.

Theory

The Flexible Authorization Framework (FAF) includes the following components (see Fig. 1):

- A *history table* whose rows describe the accesses executed.
- An *authorization table* whose rows are ►authorizations composed of the triples (s, o, $\langle sign \rangle a$), where s is the subject, o the data item, a the action and $\langle sign \rangle$ may be either "+" or "−". An authorization indicates that subject s can execute (sign +) or cannot execute (sign −) action a over authorization object o. The authorization table contains the set of explicitly specified authorizations.
- A *propagation policy*, which specifies how to obtain new derived authorizations from those explicitly stored in the authorization table (e.g., derived authorizations can be obtained according to hierarchy-based derivation policies).
- A *conflict resolution policy*, which describes how possible conflicts between the (explicit and/or derived) authorizations should be solved.
- A *decision policy*, which defines the response that should be returned to each access request. In case of conflicts or gaps (i.e., when an access is neither authorized nor denied), the decision policy determines the



Flexible Authorization Framework (FAF). Fig. 1 Functional authorization architecture

answer. In many systems, decisions assume either the open or the closed policy, where, by default, access is granted or denied, respectively.
- A set of *integrity constraints* that may impose restrictions on the content and output of the other components. Integrity rules can be used to individuate errors in the hierarchies or in the explicitly specified authorizations, or for implementing duty separation.

When a subject s requires the execution of action a on object o, the system needs to verify whether the authorization $(s, o, +a)$ or $(s, o, -a)$ can be derived using the authorization table, propagation policy, history table, conflict resolution policy, and decision policy that have been defined in the system. If a positive authorization is derived, then the access is allowed. Otherwise, if a negative authorization is derived, the access is denied.

FAF allows the representation of different propagation policies, conflict resolution policies, and decision policies that a security administrator might want to use. These policies represent only some of the possibilities and FAF is flexible enough to allow a security administrator to express what she needs for her applications. To address this issue, the functional authorization architecture can be realized through the following approach.

- The authorization table is viewed as a database.
- Policies are expressed by a restricted class of logic programs, called *authorization specification*, which have certain properties.
- The semantics of authorization specifications is given through the well-known stable model semantics and well-founded model semantics of logic programs, ensuring thus the existence of exactly one stable model.
- Accesses will be allowed or denied on the basis of the truth value of an atom associated with the access in the unique stable model.

The four decision stages (i.e., authorization table, propagation policy, conflict resolution policy, and decision policy) correspond to the following predicates:

- `Cando`$(o, s, \pm a)$ explicitly represents the authorizations defined by the security administrator: it allows (or denies, depending on the sign) subject s to execute action a on object o.
- `Dercando`$(o, s, \pm a)$ represents authorizations derived by the system according to the propagation policy.
- `Do`$(o, s, \pm a)$ represents the accesses that must be allowed or denied, and are obtained after the application of the conflict resolution and decision policies.

In addition, predicate `done` keeps the history of the accesses executed (e.g., to implement a ►Chinese Wall policy) and a predicate `error` can be used to signal errors in the specification or use of authorizations. Other predicate symbols, called `hie`-predicates, are adopted for the evaluation of hierarchical relationships between the elements of the data system (e.g., user's membership in groups, inclusion relationships between objects). A further category of predicates is the `rel`-predicates that are used to express different relationships between elements in the data system. These predicates are not fixed by the model and are application specific (e.g., `owner`(o, s) specifies that subject s is the owner of object o in the system).

Authorization specifications are stated as logic rules defined over the above predicates. To ensure stratifiability, the format of the rules is restricted as illustrated in Table 1. The stratification also reflects the different semantics given to the predicates: `cando` will be used to specify basic

Flexible Authorization Framework (FAF). Table 1 Rule composition and stratification of FAF

Stratum	Predicate	Rules defining predicates
0	`hie`-predicates `rel`-predicates `done`	Base relations Base relations Base relation
1 2	`cando` `dercando`	Body may contain `done`, `hie-`, and `rel-` literals Body may contain `cando`, `dercando`, `done`, `hie-`, and `rel-` literals. Occurrences of `dercando` literals must be positive
3	`do`	When head is of the form `do`$(\_, \_, +a)$, body may contain `cando`, `dercando`, `done`, `hie-`, and `rel-` literals
4	`do`	When head is of the form `do`$(o, s, -a)$, body contains just one literal $\neg$`do`$(o, s, +a)$
5	`error`	Body may contain `do`, `cando`, `dercando`, `done`, `hie-`, and `rel-` literals

authorizations, `dercando` will be used to enforce implication relationships and produce derived authorizations, and `do` to take the final access decision. Stratification ensures that the logic program corresponding to the rules has a unique stable model, which coincides with the well-founded semantics. Also, this model can be effectively computed in polynomial time. The authors of FAF also present a materialization technique for producing and storing the model corresponding to a set of logical rules. The model is computed on the initial specifications and updated with incremental maintenance strategies.

Open Problems

Although different logic-based models and languages have been proposed, there are still some key aspects and challenges to be addressed. Some interesting key aspects consist in enriching the expressive power of logic-based languages with the support for anonymous credentials, semantics-aware specifications, and context-aware specifications.

Recommended Reading

1. Jajodia S, Samarati P, Sapino ML, Subrahmanian VS (2001) Flexible support for multiple access control policies. ACM Trans Database Syst 26(2):214–260

Flow Blocking

►TCP Reset Injection

Forged Resets

►TCP Reset Injection

Forgery

Gerrit Bleumer
Research and Development, Francotyp Group,
Birkenwerder bei Berlin, Germany

Related Concepts

►Digital Signature; ►Non-Repudiation

Definition

The term forgery usually describes a message-related attack against a cryptographic ►digital signature scheme. That is an attack trying to fabricate a digital signature for a message without having access to the respective signer's private signing key. It is used to formally define the security requirements of digital signature schemes, in particular the requirement of *unforgeability*, which is also called ►non-repudiation.

Theory

A generally accepted formal framework for ordinary digital signatures has been given by Goldwasser et al. [2]. According to their definition a digital signature scheme consists of three algorithms: one for generating key pairs, one for signing messages, and one for verifying signatures. The key generator produces a key pair (►public key cryptography) of a signing key (private key) and a verifying key (public key). The signing algorithm takes as input a signing key and a message, and returns a signature. The verifying algorithm takes as input a verifying key, a message, and a signature, and returns a Boolean value. A signature is usually called *valid* for a message with respect to a verifying key if the verifying algorithm returns TRUE on input this verifying key, message, and signature.

The GMR definition identifies four types of attacker goals against digital signature schemes. They are in order of decreasing strength: (1) to figure out the signing key (*total break*), (2) a quasi signing algorithm that also produces signatures valid with respect to the victim's verifying key (*universal forgery*), (3) to find a signature for a new message selected by the attacker (►selective forgery), or (4) to find a signature for any one new message (►existential forgery).

They further define *passive attacks* and *active attacks* (►cryptanalysis) for cryptographic signature schemes. An attack is called passive if it is restricted to the access of the verifying key and a number of signed messages (*known message attack*). An attack is called active if it also accesses the signer to ask for signatures on messages chosen by the attacker (*chosen message attack*). The attack is successful if the attacker can come up with a signature for a new message, i.e., one for which the signer has not provided a signature before. A stronger active attack is where each message may be chosen by taking into account the signer's responses to all previously chosen messages (*adaptive chosen message attack*). Orthogonal to the classification of active attacks into whether they are adaptive or not is the following classification based on how the attacker may interleave his queries to the signer [6]. In a *sequential attack*, the attacker may ask the signer to sign messages one by one. In a *concurrent attack*, the attacker may ask the signer to sign more than one message at the same time, while the attacker may interleave his queries arbitrarily.

Unforgeability of a digital signature scheme is then defined as security against a certain type of attacker goal under a certain type of attack. The type of attack defines how the attacker can interact with or has access to the honest signer (how the attacker can wring the honest signer), and the type of attacker goal defines the attacker's outcome (a new valid signature or a signing key). Clearly, the unforgeability definition of a digital signature scheme is all the stronger, the more the attacker can wring the signer and the less outcome he can expect. For instance, the ►GMR signature scheme has been proven to be secure against existential forgery under an adaptive chosen message attack [3].

There are several different types of cryptographic signatures schemes to which other or additional security requirements apply. The better known types of cryptographic signature schemes are in alphabetical order (1) ►blind signatures, (2) ►designated confirmer signatures, (3) ►fail-stop signatures, (4) ►group signatures, (5) ►threshold signatures, (6) ►undeniable signatures, (7) transitive signatures [4], and (8) monotone signatures [5]. All of them impose security requirements alternatively or in addition to the security requirements of ordinary cryptographic signature schemes described above. Most of the alternative/additional security requirements are defined by means of alternative/additional types of forgery.

In a ►blind signature scheme, the verifier does not tell the signer which message to sign, but instead "blinds" the intended message and requests a signature for the blinded message from the signer. Then the signer provides information to the verifier, from which the verifier can efficiently derive a valid signature for the intended message. The GMR framework does not apply to blind signatures because in a blind signature scheme the notion of a successful attack is undefined. If an attacker comes up with a signature for a message, it makes no sense to ask whether that message has been signed by the signer before or not, simply because the signer has no idea which messages he has signed before. Instead, two other types of active attacks have been defined for blind signature schemes: *one-more forgery* [7] and *forgery of restrictiveness* [1, 6].

A one-more forgery [7] is an attack that for some polynomially bounded integer n comes up with valid signatures for $n + 1$ pairwise different messages after the signer has provided signatures only for n messages.

A forgery of restrictiveness is an attack that comes up with a signature for a message that does not observe a predefined internal structure.

Recommended Reading

1. Brands S (1994) Untraceable off-line cash in wallet with observers. In: Stinson DR (ed) Advances in cryptology – CRYPTO'93. Lecture notes in computer science, vol 773. Springer, Berlin, pp 302–318
2. Goldwasser S, Micali S, Rivest RL (1988) A digital signature scheme secure against adaptive chosen-message attacks. SIAM J Comput 17(2):281–308
3. Menezes AJ, van Oorschoot PC, Vanstone SA (1997) Handbook of applied cryptography. CRC Press, Boca Raton, p 432
4. Micali S, Rivest R (2002) Transitive signature schemes. In: Preneel B (ed) Topics in cryptology – CT-RSA 2002. Lecture notes in computer science, vol 2271. Springer, Berlin, pp 236–243
5. Naccache D, Pointcheval D, Tymen C (2001) Monotone signatures. In: Syverson PF (ed) Financial cryptography 2001. Lecture notes in computer science, vol 2339. Springer, Berlin, pp 305–318
6. Pfitzmann B, Sadeghi A-R (1999) Coin-based anonymous fingerprinting. In: Stern J (ed) Advances in cryptology – EUROCRYPT'99. Lecture notes in computer science, vol 1592. Springer, Berlin, pp 150–164
7. Pointcheval D (1998) Strengthened security for blind signatures. In: Nyberg K (ed) Advances in cryptology – EUROCRYPT'98. Lecture notes in computer science, vol 1403. Springer, Berlin, pp 391–405

Formal Analysis of Cryptographic Protocols

CATHERINE MEADOWS
U.S. Naval Research Laboratory, Washington, DC, USA

Synonyms

Cryptographic protocol verification

Related Concepts

►Electronic Cash; ►Electronic Payment; ►Electronic Postage; ►Electronic Voting; ►Security Standards

Definition

The application of formal methods to cryptographic protocol analysis is the process of employing automated formal analysis tools, such as theorem provers or model checkers, to the problem of determining whether an attacker can prevent the protocol from accomplishing one or more of its security goals.

Background

To see the type of problem that can arise, consider the following famous example of the Needham-Schroeder public key protocol [1], and the attack discovered by Gavin Lowe [2]. The goal of this protocol is to allow A and B to secretly share two randomly generated nonces: NA generated by A and NB generated by B. The protocol uses public key encryption to achieve its goals. The protocol at the left describes the way the protocol is supposed to proceed, with KA A's key and KB B's key. The on the right describes an attack in which I is the attacker, with its own key KI.

Protocol	**Attack**
1. A -> B: KB(A,NA)	1. A -> I: KI(A,NA)
2. B -> A: KA(NA,NB)	2. I -> B: KB(A,NA)
3. A -> B: KB(NB)	3. B -> A: KA(NA,NB)
	4. A -> I: KI(NB)
	5. I -> B: KB(NB)

Thus, at the end of the attack, the attacker I is able to convince B that NB is uniquely shared between A and B, while actually it is also known by I.

There are two things to note about this attack. The first is that it depends only on a small set of atomic actions on data: encryption, decryption, concatenation, and deconcatenation. The second thing is that although the attacks are simple, they are nonintuitive and it is difficult to avoid designing protocols that are invulnerable to them. Thus cryptographic protocols are an excellent candidate for formal analysis using mechanized tools.

The first attempt to develop formal methods for the analysis of cryptographic protocols was that of Dolev and Yao [3], who developed polynomial-time algorithms for the security of a class of simple protocols. This unfortunately did not turn out to be easily extendable to more realistic protocols, but automated tools that could be used, either with or without human guidance, to search for attacks from an insecure state specified by a user, were soon to follow, such as the interrogator [4] and the NRL protocol analyzer [5]. However, formal analysis of cryptographic protocols did not attract widespread attention until the Burrows, Abadi, and Needham (BAN) logic [6] in the late 1980s. This, although it was criticized for its lack of rigor, provided a simple, intuitive method for reasoning about protocol security that also provided insight into how the protocol worked. BAN logic, instead of looking for attacks, presents rules by which principals derive conclusions about what has happened from messages they have sent and received. This general approach, although it was for a while eclipsed by the success of model checkers, is still being followed in several threads of research, although at a lower level of abstraction and a greater level of rigor, e.g., the compositional logic approach started in [7].

Interest in applying formal methods to cryptographic protocol analysis really took off in the mid-1990s however, after Lowe used the FDR model checker to find the above-mentioned attack on the Needham-Schroeder public key protocol. This demonstrated that existing formal methods tools could be applied to the problem. This resulted in the application of other model checkers to the problem, and in a new interest in developing specialized model checkers that can be applied to the problems. Probably the most widely used of these are the AVISPA tool [8], actually a suite of model-checkers, and the ProVerif tool [9], which can be used to prove not only standard protocol properties such as authentication and secrecy, but privacy properties of the sort that are found, for example, in electronic commerce and voting protocols.

The formal model most commonly used is popularly known as the Dolev–Yao model, although it is somewhat different than the model first envisioned by Dolev and Yao. In it, the crypto algebra is modeled as a free algebra. Operations on data, such as encryption and decryption, are modeled as atomic actions. The intruder is assumed to be able to read traffic, block, redirect, and modify traffic of his own. The intruder is assumed to be able to perform any operation available to a legitimate participant in the protocol. He also may be in league with a number of corrupt principals and have access to their keys. The legitimate principals execute according to the rules of the protocol, but every message they send is read by the intruder and may be modified. Thus, every message a legitimate principal receives is assumed to have been either created or passed on by the intruder.

Although the attacker has a considerable amount of freedom, the set of actions available to it can be expressed in a very regular fashion. Thus the complexity of the problem of determining security generally hinges on the number of individual executions (usually referred to as *sessions*) of protocol roles by honest principals. The problem of determining whether the intruder learns a given term, known as the secrecy problem, was proved undecidable in the unbounded session model by Durgin et al. [10] in 1999, and NP-complete in the bounded session model by Rusinowitch and Turuani [11] in 2001. Similar results have been proved for other properties, such as authentication.

Applications

Formal analysis of cryptographic protocols has been extensively applied to the analysis of standards. In some cases, their use has made substantial contributions to the design of the protocol. Examples of such uses of formal methods include analyses of the Internet Engineering Task Force's Internet Key Exchange (IKE) and Group Domain of Interpretation (GDOI) Protocols, the Kerberos protocol, and the Secure Electronic Transactions protocol, an e-commerce standard formerly supported by the major credit card companies. In several cases (e.g., proof of key possession in GDOI [12] and the public key version of

Kerberos [13]) this has led to the discovery of problems that led to a redesign of the standard.

Open Problems

The basic problem of cryptographic protocol analysis in the Dolev–Yao model is well understood, and a number of relatively mature tools are available. Thus one active area of research is exploring how well these tools can be used to analyze protocol standards. In general, these analyses have been fruitful, helping both to find flaws early on and to increase understanding of the protocol, especially when those performing the analysis can work closely with the protocol developers. Moreover, although the tools are not yet a standard part of the protocol developer's toolkit, they are beginning to be used by those who propose protocols of their own in order to bolster their security arguments.

Other research in this area has focused on improving the accuracy and scope of the tools. One approach has been to attempt to bridge the gap between formal and cryptographic proofs of security by developing systems in which a Dolev–Yao style proof implies a cryptographic proof of security. This has turned out to be a complex problem, given the considerable gap between the two types of theories, but a number of different systems have been developed for doing this, the most prominent example being the Universal Composibility model [14].

Another approach is to apply theorem-proving techniques directly to the cryptographic proofs. Although this loses the interface with the Dolev–Yao theory, it has the advantage that one can apply formal methods directly to the cryptographic argument. This approach has been gaining popularity as tools and methods become more available. The most prominent example is CryptoVerif [15], which provides automated support for game-based cryptographic proofs.

A third approach is to extend the Dolev–Yao model by introducing equational theories obeyed by the functions used in a protocol. For example, modular exponentiation is associative and commutative, and concatenation is associative. This presents a middle ground between the Dolev–Yao model and the more expressive cryptographic models: it is more expressive than the Dolev–Yao model, but it is still possible to use model checkers to verify the cryptographic protocols. Maude-NPA [16] is approaching this problem in the most comprehensive manner, but other tools, such as ProVerif and AVISPA, also provide support for various theories.

Recommended Reading

1. Needham RM, Schroeder MD (1978) Using encryption for authentication in large networks of computers. Commun ACM 21(12):993–999
2. Lowe G (1996) Breaking and fixing the Needham-Schroeder Public-Key Protocol using FDR. In Proceedings of TACAS '96. Springer, Berlin, pp 147–166
3. Dolev D, Yao A (1983) On the security of public key protocols. IEEE Trans Inf Theory 29(2):198–208
4. Millen JK, Clark SC, Freedman SB (1987) The interrogator: protocol security analysis. IEEE Trans Softw Eng, SE-13(2): 274–288
5. Meadows C (1992) Applying formal methods to the analysis of a key management protocol. J Comput Secur 1(1):5–36
6. Burrows M, Abadi M, Needham R (1990) A logic of authentication. ACM Trans Comput Syst 8(1):18–36
7. Durgin NA, Mitchell JC, Pavlovic D (2003) A compositional logic for proving security properties of protocols. J Comput Secur 11(4):677–721
8. Armando A, Basin D, Boichut Y, Chevalier Y, Compagna L, Cuellar J, Drielsma P, Heam P, Kouchnarenko O, Mantovini J, Modersheim S, Von Hoheimb D, Rusinowitch M, Santiago J, Turuani M, Vigano L, Vigneron L (2005) The AVISPA Tool for the Automatic Validation of Internet Security Protocols and Applications. In: Proceedings of CAV 2005. Springer
9. Blanchet B (2005) An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In: 20th international conference on automated deduction (CADE-20), Tallinn
10. Durgin NA, Lincoln PD, Mitchell JC, Scedrov A (1999) Undecidability of bounded security protocols. In: Workshop on formal methods and security protocols (FMSP'99), Trento
11. Rusinowitch M, Turuani M (2001) Protocol insecurity with finite number of sessions is NP-complete. In: Proceedings of computer security foundations workshop
12. Meadows C, Pavlovic D (2005) Deriving, attacking, and defending the GDOI protocol. In: ESORICS 2005. Springer
13. Cervesato I, Jaggard AD, Scedrov A, Tsay J-K, Walstad C (2008) Breaking and fixing public-key kerberos, extended abstract. In: Okada M, Satoh I (eds) Advances in computer science – ASIAN 2006, Tokyo, Dec 2006. Springer LNCS, vol 4435, Springer
14. Backes M, Pfitzmann B (2004) A cryptographically sound security proof of the Needham-Schroeder-Lowe Public-Key Protocol. IEEE J Sel Area Comput (JSAC) 22(10):2075–2086
15. Blanchet B (2008) A computationally sound mechanized prover for security protocols. IEEE Trans Dependable Secur Comput 5(4):193–207
16. Escobar S, Meadows C, Meseguer J (2009) Maude-NPA: cryptographic protocol analysis modulo equational properties. FOSAD 2007/2008/2009 tutorial lectures, LNCS, vol 5705. Springer, pp 1–50

Formal Analysis of Security APIs

Graham Steel
Laboratoire Specification et Verification, INRIA, CNRS & ENS, Cachan, France

Related Concepts

►Security Protocols Analysis

Definition

An application program interface (API) is called a security API if it is designed so that no matter what sequence of function calls are made by the possibly malicious application code, certain security properties continue to hold. Typically these interfaces arise in systems where a process running trusted code has to offer an interface to a process running untrusted code. Formal analysis of these APIs aims to establish exactly what properties are preserved in such a scenario.

Background

Longley and Rigby were the first to use formal methods to analyze a security API [3], though the term "security API" was coined later by Anderson and Bond [1].

Theory

Security API analysis typically consists of building an abstract logical model of the operations of the API functions and the capabilities of an attacker. The model allows the intruder can to call API functions and perform offline computations in any sequence, and the security properties of the interface are posed as reachability problems in the model.

Security APIs are most often studied in the context of tamper-resistant cryptographic devices, where the trusted code running inside the device offers an interface to the untrusted host machine. Any analysis of their security must therefore take into account cryptographic operations such as encryption and signing. Most security API analyses follow the so-called Dolev-Yao abstractions, where bit-strings are considered as terms in an abstract algebra, and cryptographic operations are functions on those terms, equipped with a suitable equational theory. API functions and offline computations by the malicious application are modeled as deduction rules in the algebra. A typical security property might be that certain cryptographic keys stored on the tamper-resistant device remain unknown to the attacker, which can be posed as the reachability of a state in which the intruder knows the term corresponding to this secret key. This problem is in general undecidable, in particular because security APIs can usually generate arbitrary numbers of fresh cryptographic keys and other values. This allows classical undecidable problems such as Post's correspondence problem to be encoded as secrecy problems, following results from security protocols analysis. However, for practical APIs, bounds on the amount of fresh material can often be proved to be sufficient to cover all attacks in the model. In such cases, the security problem can often be resolved with some automatic formal tool such as a model checker or theorem prover. In particular, tools used for security protocol analysis can be adapted to the problem, by considering the API as a suite of protocols, which the intruder may compose arbitrarily.

One aspect that differentiates security API analysis from security protocols analysis is that in API analysis, one encounters mutable global state: Keys may be imported and exported, and switches may be enabled and disabled according to some rules in the API. These changes will affect the behavior of the interface. This feature causes an explosion in the number of possible of states to explore, which poses a challenge to automated tools. In addition, state change may be non-monotonic, i.e., mutually exclusive choices may have to be made. Some protocol analysis tools implement optimizations assuming intruder knowledge increases monotonically. Such tools may find false attacks when used on APIs with non-monotonic state.

Applications

A major application area for security APIs is the international cash machine network, where international standards (ISO 9564/ANSI X7.8) stipulate that tamper-resistant hardware security modules (HSMs) must be used to encrypt, decrypt, and verify customers' personal identification numbers (PINs). These HSMs expose a security API to their host computers, which must be designed to keep customer PINs secure, even if a malicious application runs on the host machine. Formal analysis of security APIs has led to the discovery of many vulnerabilities in these APIs. Some involve the key management functions, which may be satisfactorily modeled in the abstract model described above. Others involve an attacker deducing the value of a PIN from the pattern of error messages arising from various carefully constructed function calls. These information leakage attacks have been analyzed in a probabilistic model with Markov decision process semantics [4].

Open Problems

An open theoretical problem in API analysis is to relate the Dolev-Yao abstractions commonly used in formal tools to the computational complexity models used in manual cryptographic analyses. This problem has been attacked in the field of protocol analysis, but in API analysis, there are additional problems caused by the mutable state shared between commands.

In practice, it is well known that the most commonly used standard for key management APIs, RSA Public Key Cryptography Standards (PKCS) #11, cannot be used to construct a secure interface for distributed key management in its current form, since it lacks a key transport format that securely preserves key metadata [2]. This permits keys to have conflicting roles, for example, key wrapping (encryption of sensitive keys) and decryption of arbitrary ciphertexts. Various alternative standards are being drafted, but the goal of a secure key management API for distributed tokens, whose security proofs have a cryptographically sound semantics, remains open at the time of writing.

Recommended Reading

1. Bond M, Anderson R (2001) API level attacks on embedded systems. IEEE Comput Magazine 34(10):67–75
2. Delaune S, Kremer S, Steel G (2008) Formal analysis of PKCS#11. In: Proceedings of the 21st IEEE Computer Security Foundations Symposium (CSF'08), Pittsburgh, PA. IEEE Computer Society Press, pp 331–344
3. Longley D, Rigby S (1992) An automatic search for security flaws in key management schemes. Comput Secur 11(1):75–89
4. Steel G (2006) Formal analysis of PIN block attacks. Theor Comput Science 367(1–2):257–270

Formal Methods

►Applications of Formal Methods to Web Application Security

Formal Methods and Access Control

Michael Huth
Department of Computing, Imperial College London, London, UK

Related Concepts

►Access-Control Architectures; ►Administrative Policies; ►Discretionary Access Control Policies (DAC); ►Formal Verification; ►Identity Management; ►Mandatory Access Control Policy (MAC); ►Multilevel Security Policies; ►Privacy-Aware Access Control Policies; ►Role-Based Access Control

Definition

A ►formal method is any technique or method that aids in the construction and validation of computer-based systems and is based, in total or in part, on rigorous mathematics. ►Access control [8] refers to any method or mechanism by which the access of principals to resources is regulated; formal methods can aid considerably in the design, validation, and implementation of access control.

Background

Access control within IT systems developed historically through simple mechanisms such as ►access-control matrices [8]. Such a matrix explicitly lists allowed access, e.g., which principal can do what actions to which objects. This approach worked well for isolated or small-scale IT systems, but is ill-suited for systems that are distributed, federated, large-scale, or complex for other reasons.

Theory and Application

To address these needs, policies emerged as a standard tool for specifying and deciding access requests. Policies have expressive mechanisms that allow for more efficient representations of requests (e.g., through roles [5]), priority composition of rules (e.g., to define and enforce overriding concerns), the inheritance of privileges (e.g., through role hierarchies [5]), etc. A policy may be phrased in a manner that is machine-readable, appreciable by system users or both.

Policies are meant to capture intended behavior of an access-control system. Mathematically, we may think of a policy as a function $p\colon \mathsf{Req} \to \mathsf{Dec}$ where Req is the set of access requests and Dec is the set of decisions made for such access requests. Two such decisions are grant (permitted access) and deny (prohibited access). But the application domain or the composition of sub-policies may require additional values in Dec. We mention conflict (rules of sub-policies provide conflicting evidence for deciding an access request), gap (the policy provides no evidence for deciding an access request), error (evaluation of the policy for an access request causes some error), halt (to stop the execution of a program), and irrelevant (to express that the policy does not apply to an access request).

The decision of composed policies should be the composition of their individual decisions. For one, this makes policy behavior independent of syntactic peculiarities. For another, distributed systems may not allow the explicit construction of a composed policy. Formal methods can then help a great deal in developing techniques for combining local policy decisions into global ones, e.g., through research on policy algebras (e.g., [2]).

Policies within IT systems are not given as mathematical functions as above, but as syntactic, perhaps distributed, objects such as a rule file that specifies a ►firewall policy. Policy analysis (e.g., the one in [6]) is therefore an important formal activity. Such analyses include conflict detection [3] (e.g., are there conflicting rules in a policy?), ►gap detection [3] (is the policy not defined for some request?), safety problem (e.g., can access privileges be confined to intended owners of such rights?), and refinement [3] (e.g., is the modified policy less permissive than the original one?).

The policy decision point of an access-control system may also have to conduct policy analysis in order to determine how a policy ought to decide on a request. Consider delegation [1], in which a principal can transfer his or her right to, or share it with, other principals (e.g., for document collaboration). A policy decision point then needs to analyze the policy and its delegation statements in order to determine if granting an access request for a principle follows directly, or is derivable through a chain of policy-compliant delegations.

Policies are expressed in policy languages (e.g., compliant with the standard ►XACML) or, indirectly, in other mathematical formalisms (e.g., in type inference systems for the enforcement of secure information flow in Java programs). The methods used in policy analysis vary greatly but depend on said concrete representation of policies. Secure information flow of a Java program might be established by showing that one can infer its type to have a policy-compliant value in a security lattice. The presence of conflict in a rule-based policy (e.g., for a ►firewall) might be established by expressing such presence in terms of a propositional constraint and then checking the satisfiability of that constraint with a SAT solver. See [6] for use of a SAT solver in this context. In contrast, role engineering [5] might use purely probabilistic methods, as familiar from artificial intelligence, to discover functional roles (e.g., ones that don't already exist as such within an organization but that reflect common access privileges) into which principals can then be abstracted.

Formal methods can therefore serve several important but distinct functions in the design, verification, and implementation of access-control systems. Role discovery, e.g., can be considered as a design activity; proving secure information flow in programs can be seen as validation of access rights where principals are programming entities such as computation threads or applets on ►smart cards; and expressing rule-based policies as propositional constraints may lead to the discovery of redundancies in policies and so improve their implementation.

Applications of formal methods also concern the administrative aspect of policy-based access control [7]. For example, a formal policy language may cleanly separate between normal access privileges (e.g., permission to collaborate on a document), administrative privileges (e.g., permission to invite others as collaborators on a document), and "super"-administrative privileges (e.g., permission to transfer ownership of a document). Engineering workable policies with such layered, administrative permissions is difficult to achieve without the aid of formal methods, which can support analysis and can provide robust administration patterns within a policy language.

Open Problems

Access control in IT systems has to be done such that people can actually use those systems. Usability [4] in the security domain studies how the use of systems and their security interact. Future formal methods may help in improving the usability of access-control systems so that private citizens and non-specialist workers alike can control the access to resources for which they are responsible.

Formal methods are also believed to be needed in the realization of robust access control in tomorrow's systems that increasingly rely on ►virtualization, software as a service, and on ►cloud computing.

Recommended Reading

1. Abadi M, Burrows M, Lampson B, Plotkin G (1993) A calculus for access control in distributed systems. In: ACM Transactions on Programming Languages and Systems, vol 15, no 4, pp 706–734
2. Rao P, Lin D, Bertino E, Li N, Lobo J (2009) An algebra for fine-grained integration of XACML policies. In: Fifteenth Symposium on Access Control Models and Technologies. ACM, pp 63–72
3. Bruns G, Dantas DS, Huth M (2007) A simple and expressive semantic framework for policy composition in access control. In: Proceedings of the Fifth Workshop on Formal Methods in Security Engineering: From Specifications to Code. ACM, pp 12–21
4. Cranor LF, Garfinkel S (2005) Security and usability. O'Reilly Media, Cambridge, MA
5. Ferraiolo DF, Kuhn DR, Chandramouli R (2007) Role-based access control, 2nd edn. Artech House, Boston, MA
6. Jeffrey A, Samak T (2009) Model checking firewall policy configurations. In: Proceedings of the 2009 IEEE International Symposium on Policies for Distributed Systems and Networks. IEEE, pp 60–67
7. Li N, Mao Z (2007) Administration in role-based access control. In: Proceedings of the Second ACM Symposium on Information, Computer and Communications Security. ACM, pp 127–138
8. Messaoud B (2006) Access control systems. Springer, New York, NY

Formal Methods for the Orange Book

Jonathan K. Millen
The MITRE Corporation, Bedford, MA, USA

Synonyms

Security verification

Related Concepts and Keywords

►Common Criteria, From a Security Policies Perspective; ►Covert Channel Analysis; ►Formal Methods and Access Control; ►Formal Methods in Certification and Evaluation; ►Orange Book; ►Reference Monitor; ►Bell-LaPadula Confidentiality Model; ►Trojan Horse

Definition

Formal methods, in the Orange Book context, were systematic methods for security analysis based on mathematics and logic, and usually supported by software verification tools.

Background

The Orange Book, or *Department of Defense Trusted Computer System Evaluation Criteria* (TCSEC), was a standard published in 1983 and updated in 1985 [4]. It was used by the National Computer Security Center (NCSC), an office of the National Security Agency, to support the technical certification of commercially available computer systems for DoD applications. An Orange Book evaluation ranked systems into six classes C1, C2, B1, B2, B3, and A1, according to their ability to protect sensitive data. Following the recommendations of the Anderson Report [1], the highest rankings, B3 and A1, had to satisfy the *reference monitor* requirements that it mediate all access of subjects to objects, be tamperproof, and be small enough to be subjected to analysis and tests. The Anderson Report also stressed that it was important to specify and implement a policy for access control.

The highest class was A1, or "Verified Design." An A1 system has a Trusted Computing Base (TCB) that implements a reference monitor. An A1 TCB was subject to several requirements for formal methods:

- A formal model of the security policy supported by the TCB, that is proven to be consistent with its axioms.
- A formal top-level specification (FTLS) that accurately describes the TCB in terms of exceptions, error messages, and effects. The FTLS must be shown consistent with the model using "a combination of formal and informal techniques."
- Formal methods must be used for a covert channel analysis.

The consistent security policy model was required for B2 and B3 as well.

The kind of model considered acceptable was a state-transition model, where transitions represented permissible changes in the access-control state resulting from TCB operations. Expectations were strongly influenced by the Bell-LaPadula model [2], which featured two invariant security properties (simple security and *-property) that defined a "secure state." These were called "axioms" in the requirement, but they stood as proof obligations for the specified state transitions and for initial state conditions. It was in this sense that a model had to be "proven consistent with its axioms." A model might, for example, define an access control matrix $A_{i,j} \subseteq \{r, w\}$ such that if $A_{i,j} = \{r\}$, then subject S_i has read-only access to object O_j.

For classes B1 and above, the security policy being modeled included *mandatory* and *discretionary* requirements. The mandatory access control policy assigned sensitivity labels to subjects and objects, and prevented a subject from having read access to an object with a higher label (simple security) or write access to an object with a lower label (*-property). Thus, for example,

$$r \in A_{i,j} \Rightarrow \sigma(S_i) \geq \sigma(O_j)$$

where σ is the sensitivity label assignment. The discretionary access control policy allowed users to limit access to named objects by named individuals.

For the purposes of these requirements, "formal" meant "mathematically precise," and a proof could, in principle, be produced manually [9]. Formal models were small enough so that their verification was practical but tedious. Formal top level specifications occupied a position midway between mathematical models and executable code. They were less abstract than a model and an order of magnitude larger, because they had to specify more detail in the operating system interface. They were nonprocedural, following Parnas's 1972 suggestion [13]. Even though they were still much smaller than actual code listings, they were large enough so that there was an incentive to seek software tool support for their expression and verification.

Roughly speaking, formal specifications were like systems of equations relating the new values of system state variables after an operation to the parameters of the operation and the prior values of all relevant state variables. These equations had conditional tests so that the effect of an operation could depend on access control constraints. Specification languages had a style similar to programming languages. An operation for a process to get read access to a file might look something like this (in no particular language):

```
get_read(proc,file) =
  if dominates(label(proc),label(file))
  then access(proc,file) := add_mode
  (access(proc,file),read)
    and return := "ok"
  else return := "simple security
   violation"
  end
```

To show that an FTLS is consistent with a model, the analyst supplied a mapping of FTLS structures, like processes and files, to model structures, like subject-object access matrices. Each operation on system structures available at the FTLS interface could then be mapped to a state change in the model. Such a state change was consistent with the model if it respected the state invariants required by the model. This would be the case if the state change could be proved equivalent to the result of a sequence of permissible model transitions.

At the time, a number of software tools were available or under development to support verification of formal specifications. The NCSC instituted an Endorsed Tools List (ETL) of systems considered mature enough to produce acceptable formal specifications and proofs [10]. In 1988, the ETL included FDM (Formal Development Methodology), featuring the Ina Jo specification language [3], and GVE, the Gypsy Verification Environment [5, 6]. Later, others were added, such as EHDM (Extended Hierarchical Development Methodology), featuring the Special specification language [12]. Models as well as FTLSs could be expressed and verified with these tools. Such tools included an automatic or interactive theorem prover capable of dealing with logical statements about system data structures.

Although some of these systems could also be used for code verification, formal code verification was not an Orange Book requirement, because it was felt to be beyond the state of practice. Code verification must prove the correctness of programs written to implement the FTLS operations. This requires a formal semantics of the programming language, which was not available for most practical implementation languages. Furthermore, TCBs were usually implemented in a combination of high-level and assembly languages.

The level of detail in an FTLS – to include exceptions and error messages – had two motivations. One was to show how the abstract secure system design expressed by the model could be refined gradually to the running code. The other motivation was the desire to support certain formal approaches to covert channel analysis [11]. Covert channels permit a Trojan horse – a malicious program acting without the user's knowledge – to communicate sensitive information to users or confederate programs at a less sensitive level. The normal access control policy prevents a program from simply copying sensitive data to a publicly readable file, but it may still be able to communicate small amounts of data at a time in devious ways, by channels not intended to carry user data. Typically, this is done by affecting shared system resources, causing state changes that become visible through error messages or timing variations.

Covert channel analysis has two steps: an identification phase and a bandwidth (information rate) analysis. Identification can be assisted by information flow tools that operate on the FTLS, such as Ina Flo (in the FDM system) and the MLS Checker (in EHDM). Flow tools made use of mandatory access control labels, extending their use to system variables, and examined each FTLS statement to ensure that the value of each system variable was not tainted by the value of any variable with higher label.

In the `get_read` example above, a flow analysis would determine that the return message depends on the label assignment array. This means that the label assignments must themselves be classified at the lowest label value, otherwise an attempt to read a higher-labeled file might give away higher-labeled information.

It was recognized that not all covert channels could be found by flow tools. In particular, an FTLS does not contain timing information necessary to find timing channels. A timing channel makes use of system clock values or other timing sources to convey information, just as Morse code uses combinations of short dots and long dashes to encode different letters. Furthermore, flow tools do not supply estimates of the bandwidth (information rate) of covert channels. Hence, formal methods had to be supplemented with code examination and information-theoretic techniques. The formal methods requirement for covert channel analysis could also be satisfied without flow analysis tools by using Kemmerer's shared resource matrix method [7], which was systematic, but took a less formal, more flexible approach that could be applied to code.

Applications

The only system that achieved the A1 class under the Orange Book was the Honeywell SCOMP (Secure Communications Processor). However, the Trusted Network Interpretation of the TCSEC (TNI, or "Red Book" [8]) carried over the Orange Book classes to network components. Mandatory or M-components, whose TCB enforced separation of multiple labeled security levels, could attain an A1 class with essentially the same use of formal methods. The Boeing MLS LAN (Multilevel Secure Local Area Network) and Gemini Trusted Network Processor accomplished this. Formal methods also had a place in later Common Criteria requirements.

Recommended Reading

1. Anderson JP (1972) Computer security technology planning study. ESDTR-73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA
2. Bell DE, La Padula LJ (1975) Secure computer system: unified exposition and multics interpretation. ESDTR-75-306. The Mitre Corporation, Bedford, MA
3. Berry DM (1985) An Ina Jo proof manager for the formal development method. ACM SIGSOFT Software Eng Notes 10(4): 19–25
4. Department of Defense (1985) Department of Defense Trusted Computer System. Evaluation Criteria. DOD 5200.28-STD, National Computer Security Center, Fort Meade, MD

5. Good DI, Akers RL, Smith LM (1988) Report on Gypsy 2.05, Computational Logic, Inc., Austin, TX
6. Kaufmann M, Young WD (1987) Comparing gypsy and the Boyer-Moore logic for specifying secure systems. In: Proceedings of the 10th National Computer Security Conference, Baltimore MD
7. Kemmerer RA (1983) Shared resource matrix methodology: an approach to identifying storage and timing channels. ACM Trans Comput Syst 1(3):256–277
8. National Computer Security Center (1987) Trusted network interpretation of the trusted computer system evaluation criteria. NCSC-TG-005
9. National Computer Security Center (1992) A guide to understanding security modeling in trusted systems. NCSC-TG-010
10. National Computer Security Center, Guidelines for formal verification systems. NCSC-TG-014-89
11. National Computer Security Center (1993) A guide to understanding covert channel analysis of trusted systems. NCSC-TG-030
12. Owre S, Rushby J, von Henke F (1991) An introduction to formal specification and verification using EHDM. SRI International. SRI-CSL-91-2
13. Parnas DL (1972) A technique for software module specification with examples. Comm ACM 15(5):330–336

Formal Methods in Certification and Evaluation

Catherine Meadows
U.S. Naval Research Laboratory, Washington, DC, USA

Synonyms

High assurance evaluation methods

Related Concepts

►Common Criteria, From a Security Policies Perspective; ►Security Standards

Definition

Before computer systems can be used for security-critical applications, they generally must be certified to be appropriate for the intended use. This process can be made easier by the evaluation of systems according to broader classes of criteria, established by a national or international standards body, which specify classes of systems with different levels of security functionality and assurance. Formal methods is an assurance method that is generally deemed appropriate for the highest levels of assurance.

Background

One of the main, indeed possibly the principal, use of formal methods in preparing computer systems for security evaluation is that formal specification languages can be used to give a precise, unambiguous specification of a computer system that can be understood by a system evaluator. At the very highest level of assurance, these are included with formal proofs of security, via a formal proof of correspondence between a formal security model (that is, a formal description of the security properties of the system) and formal specification of the security-relevant portions of the system. This general approach was introduced in the US TCSEC (also known as the *Orange Book*) [1], which was the first such set of criteria. It was subsequently adopted, with some changes, by criteria of other countries, including the European ITSEC [2], and finally by the *Common Criteria* [3], which replaces these earlier criteria and has been adopted by a number of countries in North America and Europe, as well as Australia. The main difference between the Common Criteria and the TCSEC, as far as the use of formal methods is concerned, is that the TCSEC levels applied to both assurance and functionality, thus enforcing a tight coupling between the two, while the Common Criteria levels apply only to assurance. The Common Criteria requires that desired security functionality be specified in separate "Target of Evaluation" documents.

The Common Criteria application levels range from EAL1 to EAL7, with formal methods being introduced at EAL5. EAL5 requires the use of semiformal design description, where "semiformal" means that the specification has "a restricted syntax and a defined semantics." EAL6 requires a formal model of the security policy plus a semiformal functional specification, where "formal" means a "restricted syntax and a defined semantics based on well-established mathematical concepts." EAL7 requires a formal model of the security policy and a formal functional specification, along with a formal verification of correspondence between the two. Although mechanized proofs are not explicitly required for EAL7, the advantage of using them is noted and generally, products that supply formal verification will rely at least partially on such mechanized proofs.

It is also possible to certify systems at EALX+, meaning the system satisfies all of the requirements of EALX, and some of the requirements of EALX + 1. For EAL6+, this usually means that some formal verification is done, although not enough to support certification at EAL7.

Since certifications are usually done on commercial products, details the ways in which formal methods are used in such certifications are proprietary. However, there are some papers available that give informative accounts that give the reader an understanding of the type of verification that is required for the evaluation process. In particular, Heitmeyer et al. [4] describe the formal verification of an embedded security device intended for evaluation against the Common Criteria, while Woodcock et al. [5] describe the certification of the Mondex electronic purse to ITSEC Level E6, a level similar to EAL7.

Applications and Experiments

Currently (November 2010), the Common Criteria Certified Products list [6] gives over 100 systems certified at EAL5 or EAL5+. EAL6 and higher evaluated systems are much more rare, however. Currently, the Common Criteria Certified Products list gives only one such system: the Green Hills Software INTEGRITY-178B Separation Kernel, certified at EAL6+. Some of the national certification organizations list additional products: for example, the French Agence Nationale de la Sécurité des Systèmes d'Information lists five microcontrollers certified at EAL6+ [7]. Other products are in the pipeline. For example, the US NIAP Common Criteria Evaluation and Validation Scheme for IT Security (CCEVS) lists two: the Boeing Secure Network Server (EAL7) and the Wind River VxWorks MILS operating system (EAL6+) [8].

These figures show that the evaluation of systems at the higher assurance levels that require the use of formal methods is not only within the state of the art, but it is seen as a viable business opportunity for the developers of systems that fill a certain type of niche: that is, systems such as separation kernels, network servers, and microcontrollers that can provide security functionality to other systems that make use of them. These systems are relatively simple, which keeps the costs of verification down, and because they can be used by many other systems, the costs are amortized. However, the fact that the number of these systems is small, and most of that small number to not attempt EAL7, shows that full-scale verification in support of security evaluation is still a challenging problem.

Open Problems

The main problem faced when undergoing high assurance evaluation is the time and expense. The additional work in specifying and verifying the product, as well as reviewing the artifacts produced by the verifiers, adds to the cost of developing the product, while the additional time not only adds to the costs, but incurs the risk that a system may be obsolete by the time it is certified. Thus, research is needed in making formal methods more efficient, easier to use, and more agile. Areas of interest include methods for incremental verification (so that small changes to a system require only a small amount of re-verification), improved methods for code verification, and methods for extracting formal specifications from code.

Recommended Reading

1. DoD trusted computer security evaluation criteria, Dod 5200.28-STD, December 1985
2. Information technology security evaluation criteria (ITSEC): preliminary harmonised criteria, Document COM(90) 314, Version 1.2. Commission of the European Communities, June 1991
3. Common criteria for information technology security evaluation, Version 3.1, Revision 3, CCMB-2009-07-01, CMMB-2009-07-02, CCMB-2009-07-03, July 2009
4. Heitmeyer C, Archer M, Leonard E, McLean J (2008) Applying formal methods to a certifiably secure software system. IEEE Trans Software Eng 34(1):82–88
5. Woodcock J, Stepney S, Cooper D, Clark J, Jacob J (2008) The certification of the Mondex electronic purse to ITSEC level E6. Form Asp Comp 20(1):5–19
6. Common criteria certified products list, common criteria portal. http://www.commoncriteriaportal.org/products/
7. ANSSI certified products list. http://www.ssi.gouv.fr/site_rubrique101.html
8. NIAP CCEVS: products and protection profiles in evaluation. http://www.niap-ccevs.org/cc-scheme/in_evaluation/

Forward Secrecy

►Perfect Forward Secrecy

FPGA Field Programmable Gate Array

►Trusted Computing

FPGAs in Cryptography

Tim E. Güneysu
Department of Electrical Engineering and Information Technology, Ruhr-University Bochum, Bochum, Germany

Synonyms

Cryptography on reconfigurable devices

Definition

Field Programmable Gate Arrays (FPGA) are generic hardware devices that can dynamically reconfigure their internal structure to perform arbitrary logic functions. In this way, they combine the advantage of efficient hardware implementations with the flexibility of software programs.

This unique property is particularly useful for computationally challenging cryptosystems that benefit from the high performance in hardware implementations, while still preserving the option for security updates in case the security system was compromised.

Background

Field Programmable Gate Arrays were invented in 1985 and originally targeted applications in the telecommunication sector. To maintain the feature of reconfigurability, FPGAs consist of a large, two-dimensional array of configurable logic elements (CLE) that are interconnected with a programmable switch matrix. The CLEs can be individually configured to perform an atomic logic function with a specific number of inputs and outputs. However, due to limitations in technology, most FPGAs only allow a volatile configuration of the CLE. In other words, the entire setup of the FPGA device is lost on power-down and needs to be reloaded after each system start-up. Fortunately, this also allows a straightforward exchange of the hardware application on the FPGA by simply replacing this initial device configuration, which is stored as an external bit file.

With the evolution of FPGA devices with respect to their logic density, the first cryptographic implementations on an FPGA became possible in the early 1990. In this context, low-level bit operations of cryptographic schemes can be directly mapped onto CLE functions, which is usually considered as the main advantage of cryptographic implementations on FPGAs with respect to software implementations. Similarly, bit permutations can instantly be performed by just selecting the appropriate routes of the programmable switch matrix. In software, such bit-level operations often require a significant amount of clock cycles for masking and shifting the individual bits in a register. In particular, block and stream ciphers, as well as asymmetric cryptosystems based on elliptic and hyperelliptic curves over binary fields, involve many of such simple bit-level operations and can thus be very efficiently implemented on FPGAs.

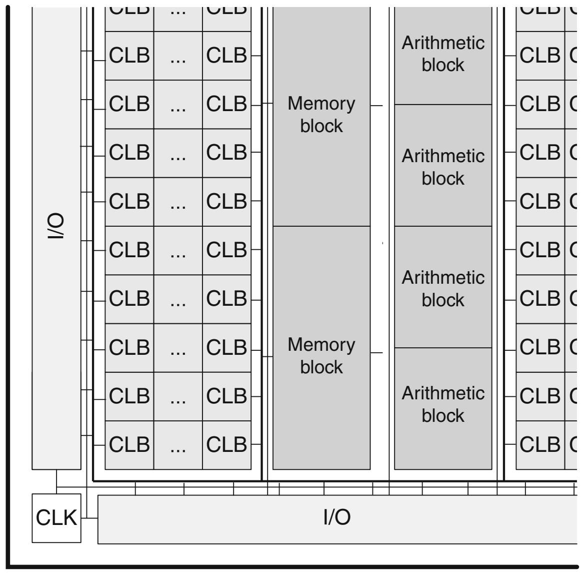
In the last years, FPGA vendors decided to add further components to FPGA architectures. They found that just a large amount of generic CLEs is not sufficient for all demands of an application. To satisfy the need for continuous storage, they integrated dedicated memory blocks, each providing a few kilobits of storage. In addition to that, they added arithmetic blocks that can accelerate single-precision integer multiplications and additions using dedicated circuits (instead of implementing them with generic CLEs). These later extensions are extremely beneficial for cryptographic implementations: Memory blocks are often used to store lookup values required by cryptographic schemes, e.g., to implement a nonlinear substitution-table lookup of block ciphers. The dedicated arithmetic blocks are useful when implementing cryptosystems involving multi-precision modular arithmetic. As each multi-precision operation consists of a large amount of single-precision computations, such a computation can be significantly accelerated by employing fast arithmetic blocks in the FPGA to compute the single-precision results.

Figure 1 shows the common structure that is shared by most recently used FPGA devices. Note that in addition to CLEs, dedicated memory and arithmetic blocks, these devices contain further components for clock generation and distribution, as well as logic to implement I/O interfaces.

Applications

In practice, FPGAs can be used to implement nearly all cryptographic algorithms. However, whether the FPGA is the optimal choice to implement a specific cryptographic system is mainly dependent on four highly important properties of the application:

- *Based on Low-level Operations*: If a cryptosystem defines the use of many basic bit-level operations, the FPGA can place these operations very efficiently in CLBs. This is usually by far more efficient than bit operations based on generic instructions available on typical microprocessors.



FPGAs in Cryptography. Fig. 1 Architecture of a recent FPGA device

- *Suitable for High Parallelism*: If individual parts of an (cryptographic) application can be parallelized, multiple computation units can be instantiated on the same FPGA to execute each of those parts concurrently. For n such computation units, this can optimally lead to an n-fold performance speedup compared to a single-threaded software implementation.
- *Uniform in Data Flow*: For cryptosystems that process data in uniform operation without any conditional computations or interrupts, advanced design techniques such as pipelining can be used in an FPGA design to significantly increase the data throughput. Pipelining allows to process different data streams in the same clock cycle – each at a different stage of the algorithm – similarly to a manufacturing process using an assembly line. Note, however, that any irregularity in the data flow is likely to cause a stall in the pipeline and demands a costly individual processing – leading to significantly decreased efficiency.
- *Constraint on Memory*: FPGAs usually provide only a few megabit of fast on-chip storage, dependent on the number of memory blocks they contain. Hence, if a cryptographic application requires more memory than provided by the FPGA, the system designer needs to install external memory on the FPGA board. In this case, the access to external memory requires an additional memory controller, which in turn serializes all individual accesses to the external storage. Therefore, even with two or more parallel memory controllers (and corresponding memory module on the board), applications involving a large number of parallel processors on the same FPGA are likely to starve from data shortage due to the serial access to the external memory.

An application perfectly suited for FPGAs (i.e., fulfilling all the mentioned criteria above) is the standardized Data Encryption Standard (DES) block cipher. DES was specifically designed for optimal hardware efficiency and thus uses only straightforward low-level bit operations and only a negligible amount of memory. A pipelined implementation of DES based on four separate processors on a low-cost Xilinx Spartan-3 FPGA (currently approx. US$20–30) can provide 500 million encryptions per second at 125 MHz. On the contrary, a single-threaded software implementation of DES on an Intel Pentium 4 running at 3 GHz (Prescott) computes roughly two million encryptions per second. This demonstrates the impressive cost–performance advantage that FPGAs can gain with respect to software solutions for specific designs.

Besides constructive use cases, this performance advantage in specific cryptographic operations can be also very useful for ►special-purpose cryptanalytical hardware. For example, an FPGA-based cluster system (cf. cryptanalysis with COPACOBANA in recommended reading) can perform an exhaustive key search on DES in less than a week (average time) with 120 simultaneously running FPGA devices.

Open Problems

Complex applications and problems that have an inherently serial nature (such as many cryptographic hash functions) are usually less efficiently implemented on FPGAs compared to software-based solutions. Since recent microprocessors operate at clock frequencies of several gigahertz and provide multiple cores on the same chip, they can perform inherently serial computations faster than FPGAs with a maximum device frequency of only a few hundred megahertz. With respect to this, achieving similarly high frequencies on an FPGA is impossible due the overhead introduced by the flexible reconfiguration feature. This actually can only be remedied by a radically new CLB design with significantly less overhead.

Recommended Reading

1. Rodríguez-Henríquez F, Saqib NA, Díaz-Pèrez A, Koc CK (2006) Cryptographic algorithms on reconfigurable hardware. Springer, Berlin. ISBN: 978-0-387-33883-5
2. Güneysu T, Kasper T, Novotny M, Paar C, Rupp A (2008) Cryptanalysis with COPACOBANA. IEEE Trans Comput; IEEE Comput Soc 57(11):1498–1513

Function Field Sieve

Emmanuel Thome
INRIA Lorraine Campus Scientifique,
VILLERS-LÈS-NANCY CEDEX, France

Related Concepts

►Discrete Logarithm Problem; ►Generic Attacks Against DLP; ►Index Calculus Attack; ►Number Field Sieve for the DLP

Definition

The *Function Field Sieve* (FFS) is an algorithm originally due to Adleman [1, 2] for solving the ►Discrete Logarithm Problem in (the multiplicative group of) ►finite fields of small characteristic.

Background

The Function Field Sieve can be used to compute discrete logarithms in $\mathrm{GF}(p^k)^*$ as long as k grows at least as fast as $(\log p)^2$. Under these conditions, the complexity of the function field sieve with respect to p and k is given by the expression in ►*L-notation*

$$L_{p^k}\left(1/3, (32/9)^{1/3}\right) = \exp\left((32/9)^{1/3}(k \log p)^{1/3} (\log(k \log p))^{2/3} \cdot (1 + o(1))\right),$$

thereby achieving the best known complexity to date for computing discrete logarithms in such finite fields (in full generality).

The Function Field Sieve is very similar to the Number Field Sieve (the latter, best known as an algorithm for ►integer factoring, also exists as an algorithm to compute discrete logarithms in finite fields $\mathrm{GF}(p^k)$ when k grows *slower* than $(\log p)^2$ ►Number Field Sieve for Factoring and ►Number Field Sieve for DLP). The Function Field Sieve's earliest predecessor is Coppersmith's algorithm (1982), which was the very first algorithm of complexity $L(1/3)$ for solving discrete logarithms in the field $\mathrm{GF}(2^n)$.

Theory

It is assumed that a defining polynomial $f(x)$ for the finite field $\mathrm{GF}(p^k)$ has been chosen, as well as the equation of an absolutely irreducible plane curve $h(x,y)$. Another input is a polynomial $m(x)$ which satisfies $h(x,m(x)) \equiv 0 \bmod f(x)$. The Function Field Sieve looks at bivariate polynomials $\phi = a(x) - yb(x)$ from two different perspectives. On the one hand, reducing ϕ modulo $y - m(x)$ yields a polynomial in x. Reducing this polynomial in turn modulo $f(x)$ yields a finite field element. This is called the "rational side." On the other hand, one may consider ϕ modulo h, thereby considering it as a function on the curve defined by h. Evaluating this function at the place given by $(f(x), y - m(x))$ gives again the same finite field element. This is called the "algebraic side."

Functions meeting the *smoothness* condition on both sides are of interest to the Function Field Sieve. On the rational side, smoothness of $a(x) - m(x)b(x)$ is sought. On the algebraic side, one requires that the divisor of the function ϕ be smooth, in that only prime divisors from a predefined set (the algebraic *factor base*) appear in its support. This condition is checked by the factorization of the norm of ϕ as an element of an algebraic extension of $k(x)$. Once one has collected a sufficiently large number of functions which satisfy both smoothness conditions, it is possible to obtain the logarithm of the finite field elements corresponding to the factor base elements using linear algebra over $\mathbb{Z}/N\mathbb{Z}$, where N is $p^k - 1$ (or possibly the cardinality of the prime subgroup of interest within this field).

Furthermore, logarithms of arbitrary finite field elements can be obtained from a process known as *special-q* descent, as described by Coppersmith [3]. An adaptation of this descent to the FFS case is given by Enge, Gaudry, and Thomé [4].

Technical obstructions to the function field sieve are quite similar to those encountered in the number field sieve. However some may be overcome in an easier way. In particular it is interesting to choose the curve h as what is known as a C_{ab} curve. This ensures that only one infinite place exists in the function field. Therefore, no obstruction stems from the "unit group" (whose role is played by places at infinity). Even with this convenience condition, the flexibility in choosing the polynomial h as well as the polynomial f defining the finite field is immense due to the fact that all finite fields having the same cardinality are isomorphic. This flexibility accounts for the good complexity of the function field sieve.

Experimental Results

Practical experiments with the function field sieve began with Joux and Lercier's computation of discrete logarithm in $\mathrm{GF}(2^{521})$ in 2001 [5], and the present record is a computation of discrete logarithms in $\mathrm{GF}(2^{613})$ by the same authors in 2005.

Recommended Reading

1. Adleman LM (1994) The function field sieve. In: Adleman LM, Huang M-D (eds) ANTS-I: 1st Algorithmic Number Theory Symposium, Cornell University, Ithaca, 6–9 May 1994. Lecture notes in computer science, vol 877. Springer, Berlin, pp 108–121
2. Adleman LM, Huang M-D (1999) Function field sieve methods for discrete logarithms over finite fields. Inform Comput 151(1):5–16
3. Coppersmith D (1984) Fast evaluation of logarithms in fields of characteristic two. IEEE Trans Inform Theory IT–30(4):587–594
4. Enge A, Gaudry P, Thomé E (2011) An L(1/3) discrete logarithm algorithm for low degree curves. J cryptol 24(1):24–41
5. Joux A, Lercier R (2002) The function field sieve is quite special. In: Fieker C, Kohel DR (eds) ANTS-V: 5th Algorithmic Number Theory Symposium, Sydney, 7–12 July 2002. Lecture notes in computer science, vol 2369. Springer, Berlin, pp 431–445