(5) Cryptography and Security Technology

Craig Gentry and Zulfikar Ramzan

This article discusses new cryptographic algorithms and protocols designed to enable DoCoMo to be a trust broker in a large heterogeneous network.

1. Introduction

To support DoCoMo's Fourth-Generation (4G) vision, any security solution must be exceptionally scalable and flexible. Not only will DoCoMo's client base be huge, but client networking environments may be heterogeneous and constantly changing. There will also be diversity in how clients will use the system, with some wishing to access private information in a secure fashion and others wishing to conduct secure transactions with third-party service providers, for example. In the 4G network, DoCoMo's challenge—indeed, its opportunity—is to find a security solution that promotes growth and flexibility rather than inhibiting them.

In this article, we provide a snapshot of our current research into cryptographic algorithms that are designed to allow DoCoMo to provide security services to a large client base in diverse environments. We begin by describing three techniques designed to simplify DoCoMo's management of a "Public Key Infrastructure" (PKI)—namely, Hierarchical Identity-Based Cryptography (HIBC), certificateless Public-Key Cryptography (PKC), and aggregate signatures. Next, we describe two techniques designed to enable secure content distribution: an exceptionally efficient microcredit scheme using a new type of hash tree, and a stream authentication scheme that allows an intermediate proxy to transcode the stream dynamically without breaking end-to-end security.

2. Simplifying the PKI

PKC allows two parties with no direct trust relationship to communicate securely, without involving a Trusted Third Party (TTP) in the communication. Thus, PKC is more flexible than symmetric cryptography. However, before communication can begin, public keys must be distributed and authenticated. For example, before sending a message encrypted with a recipient's public key, the sender must obtain the recipient's public key and must receive assurance, from an entity that it trusts, that the public key is authentic—i.e., it really belongs to the recipient (and not someone else).

Distributing and authenticating public keys typically requires a PKI. The PKI usually includes a TTP, called a Certificate Authority (CA) that generates public-key certificates; each certificate is a digital signature by the CA that securely binds together several quantities usually including at least the name of a user U and his public key PK_U. Often, the CA includes a serial number SN_U (to simplify its management of the certificates), as well as the certificate's issue date I_U and expiration date E_U. By issuing the signature $Sig_{CA}(U, PK_U, SN_U, I_U, E_U)$, the CA attests to its belief that PK_U is (and will be) user U's authentic public key from the current date I_U to the future date E_U.

Since CAs cannot predict the future, circumstances may require a certificate to be revoked before its intended expiration date; e.g., if a user's secret key is accidentally revealed or compromised, the user himself may wish to revoke his certificate. Alternatively, the user's company may request revocation if the user leaves the company or changes position and is no longer entitled to use the key. If a certificate is revocable, then third parties should not rely on that certificate unless the CA distributes certificate status information indicating whether the certificate is currently valid. This certificate status information must be fresh—e.g., to within a day—and it must be widely distributed to all parties relying on the current validity of the user's public key. The task of distributing large amounts of fresh certification information is known as the "certificate revocation problem."

The most well-known approach to the certificate revocation problem is the Certificate Revocation List (CRL). A CRL is simply a list of certificates that have been revoked before their expiration dates. The CA issues this list periodically, with the CA's signature affixed. Since the CA will likely revoke many of its certificates—say, 10% if they are issued with an intended validity period of one year[1]—the CRL will be quite long if the CA has many clients. Nonetheless, the complete list must be transmitted to any party that wants to perform a certificate sta-

tus check. There are refinements to this approach, such as Δ-CRLs that list only those certificates that have been revoked since the CA's last update; however, the transmission costs and the associated infrastructure costs are still quite high.

Another proposal is called the Online Certificate Status Protocol (OCSP). The CA responds to a certificate status query by generating (online) a signed report of the certificate's current status. This reduces transmission costs to a single signature per query, but it substantially increases computation costs. It also decreases security: if the CA is centralized, it becomes highly vulnerable to Denial of Service (DoS) attacks; if it is distributed and each server has its own private key, then compromising any server compromises the entire system [1].

In 1984, Shamir [2] described an approach different from PKI—called Identity-Based Cryptography (IBC)—designed to eliminate certificates altogether. In IBC, a user's public key is derived directly from aspects of his identity ID_U—e.g., his IP address or e-mail address. The user's private key PrK_U is generated by a TTP, called a Private Key Generator (PKG). The PKG has a public key PK_{PKG} and a "master" private key PrK_{PKG}; it computes PrK_U from PrK_{PKG} and ID_U. In IBC, public keys do not need to be distributed; i.e., as long as a sender knows PK_{PKG} and ID_U, it can encrypt a message to U. Moreover, certificates are unnecessary since U can only decrypt if he has received a private key from the PKG. Although Shamir described an identity-based signature scheme, identity-based encryption schemes were developed only recently [3,4]. HIBC and certificateless PKC, described below, not only eliminate the need for certificates, but improve upon basic IBC in different ways.

2.1 Hierarchical Identity-Based Cryptography

IBC eliminates the need to distribute user's public keys, but if U_1 uses a different PKG from U_2, then U_1 must obtain the public key of U_2's PKG before encrypting a message to U_2. It is therefore preferable to minimize the number of different PKGs. On the other hand, it is infeasible for a single PKG to handle private-key generation for a very large number of users, because of the major scalability problem associated with the processing for generating the private keys. HIBC solves this dilemma by allowing a hierarchical topology in which a root PKG may delegate private-key generation to lower-level PKGs, while outside users only need to obtain the public key PK_{root} of the root PKG. HIBC, like the Boneh-Franklin IBC scheme, uses elliptic curves and a mathematical construct called a "pairing" [5]. Here, we describe the HIBC approach briefly, with the aid of specific examples.

In HIBC, each user has an ID-tuple that represents his position in the hierarchy. For example, if Alice's e-mail address is alice@cs.univ.edu, then her ID-tuple may be (edu, univ, cs, alice). Alice's parent in the hierarchy is the client with ID-tuple (edu, univ, cs); this client might be managed by the system administrator in her university's Computer Science (CS) department.

The root PKG uses its master private key to generate the private keys of the PKGs that are on the level immediately below it in the hierarchy. In this example, the system administrator obtains a private key for the CS department from the university, which makes it possible for Alice to then obtain her private key from the CS department. To accomplish this, Alice gives her ID-tuple to the CS department, along with proof of identity. After verifying her identity, the CS department computes Alice's private key as a function of its own private key and Alice's ID-tuple (and possibly additional widely-available information, such as the current date). The CS department sends Alice her private key via a secure channel. Since Alice's CS department is "local," proving her identity and obtaining a secure channel is easier than in non-hierarchical IBC. The CS department's ancestors in the hierarchy do not directly participate in the generation of Alice's private key.

Using hierarchical identity-based encryption as an example (although hierarchical identity-based signatures are also possible), another person, let's call him Bob, can encrypt a message to Alice as long as he knows PK_{root} and her ID-tuple (and possibly additional information for validity management such as the date). Bob does not need to obtain any public keys belonging to Alice, or to her ancestors in the hierarchy below the root, because they do not have public keys. He also does not need to obtain Alice's up-to-date certificate, because he knows that Alice will not be able to decrypt the message unless she has an up-to-date private key. Bob combines PK_{root} and Alice's ID-tuple with his message M to generate a ciphertext C. Alice decrypts C using her private key. The encryption time, decryption time and ciphertext length are all proportional to the recipient's (Alice's) depth in the hierarchy, but this will typically be a fairly small number.

An interesting property of HIBC is that, since private key generation is hierarchical, all of the recipient's ancestors in the hierarchy can also decrypt the message. This "key escrow" property may be desirable in some settings, but there are many

others in which it is not. Below, we present a certificateless PKC that eliminates the key escrow property while retaining the benefit of IBC and HIBC of not requiring certificates.

2.2 Certificateless Public-Key Cryptography

A Certificateless Public-Key Encryption (CPKE) scheme [6][*1] is similar to a traditional (non-identity-based) public-key encryption scheme in that a recipient generates its own public-key/private-key pair and furnishes the public key and a proof of identity to a TTP to be "certified." (Certificateless signatures are also possible when aggregate signatures are used as described below.) However, instead of generating traditional certificates, the TTP generates a second decryption key for the recipient like a PKG does in an identity-based encryption scheme. In essence, a sender "doubly encrypts" its message—once using the recipient's personal public key in a traditional public-key encryption scheme, and once using the TTP's public key and the recipient's identifying information in an identity-based encryption scheme. The recipient needs both her personal private key and an up-to-date decryption key from the TTP to decrypt.

The sender does not need to check the recipient's certificate status, since the sender knows that the recipient will not be able to decrypt unless the recipient's TTP has given it a current decryption key—in effect, "certifying" the recipient. The TTP does not have key escrow; it cannot decrypt because it does not know the recipient's personal private key. Moreover, as long as the recipient's personal private key is protected, the TTP can transmit the recipient's identity-based decryption key in the clear.

In CPKE, the sender must obtain the recipient's public key, but this can be viewed as a one-time cost because the recipient's public key should not change very often. Eliminating certificate status queries also allows the TTP to reduce its infrastructure. Unlike traditional PKI, where the TTP must respond to certificate status queries that may come from senders anywhere in the network and may concern any recipient, a TTP that uses CPKE merely needs to transmit decryption keys to its clients (the recipients). Having the TTP only deal with clients is more attractive from a business model perspective. Also, the decryption keys could even be "pushed" to clients, and—in "incremental" CPKE, as described in [6]—multicast might be used to dramatically reduce the CA's transmission costs. This push model makes the TTP less susceptible to DoS attacks.

Using incremental CPKE, a TTP can handle a huge number of clients very efficiently, even if the TTP updates client decryption keys very frequently (e.g., hourly). The TTP only needs to compute $R_{period}(\log(N/R_{period}))$ decryption keys per time period, where R_{period} is the number of clients whose public keys have become invalid during the time period, and N is the total number of clients. Setting N to be 250 million, and assuming a 10% revocation rate per year, R_{hour}= (250 million)/(10×365×24), which is approximately 2850.

Using this value of R_{hour}, we find that the TTP only needs to compute about 13 decryption keys per second. A 1 GHz Pentium III processor can compute about 280 (elliptic-curve-based) decryption keys per second; so the TTP's computational overhead is quite reasonable. Full details, and additional analysis of computational and transmission requirements, are provided in [6].

In summary, CPKE allows a TTP, such as DoCoMo, to dramatically reduce the infrastructure, as well as the computation and transmission overhead, needed to manage public keys for a huge number of users.

2.3 Aggregate Signatures

An aggregate signature scheme, first introduced in [8], allows multiple digital signatures on multiple (possibly distinct) messages by multiple signers to be compressed down to the bit-length of a single short signature. For example, in a PKI of depth t, a user may have a certificate chain consisting of t certificates, where each certificate is the signature of a parent in the hierarchy certifying the public key of its child. A signature scheme permitting aggregation thus reduces the bandwidth overhead involved in PKI, which may be particularly useful in bandwidth-constrained environments.

Interestingly, certain types of aggregate signatures, such as those described in [8], may also be used as decryption keys. A sender (using CPKE) may thus make a recipient's ability to decrypt contingent on its possession of its personal private key and an aggregate signature containing the signatures of specified signers on specified documents. This concept of conditioning a recipient's ability to decrypt on the "authorization" of one or more specified entities may have applications beyond simplifying PKI.

*1 In [6], the concept is called "Certificate-Based Encryption (CBE)", but the term "certificateless", subsequently used in [7], may better capture the revolutionary nature of the idea.

3. Cryptography Tailored for Specific Applications

3.1 Microcredits for Verifiable Foreign Service Provider Metering

With the explosive growth of mobile communications, users may frequently access value-added services through Foreign Service Provider (FSP)s. These providers will interact directly with users, later providing details to a user's Home Service Provider (HSP) regarding the services rendered; the HSP, in turn, bills the user. One critical concern is that the FSP might inflate the usage figures it furnishes to the HSP.

We are addressing this issue with a microcredit scheme. When a user employs a service, small tokens, called microcredits, are issued to serve as the basis for later calculation of charges. This scheme is efficient in that the verification time required by the HSP is only logarithmic relative the number of microcredit transactions, and the verification time required by the FSP is a constant. Furthermore, the cost of communication between user and FSP for issuing tokens is also a constant.

The basic idea behind the scheme is as follows. Each user generates a number of "microcredit tokens." In our scheme, the tokens are small (40 bytes on average) and relatively inexpensive to generate; for example, about 5000 microcredits can be generated in 4.2 ms on a 2.1 GHz Pentium IV.[*2] The user signs the tokens with its private signature key so that these tokens are uniquely tied to that user. Only one such signature is necessary per batch of microcredits. In its interactions with the FSP, the user may provide a microcredit token at each well-defined interval (e.g., with each packet or at every five seconds). The FSP has to perform one cryptographic hash function computation to verify each microcredit. If service is terminated after t intervals, the FSP sends the tth microcredit token, together with $O(\log t)$ additional values that are needed to validate the token, to the HSP. The HSP can validate the tth microcredit in $O(\log t)$ time. If it is valid, it bills the user for t intervals of the FSP's service.

To construct m microcredits using a QuasiModo tree (an improvement upon traditional Merkle trees), the user performs the following steps. It first generates m+1 20-byte values at random. It labels the leaves of a (nearly) balanced binary tree with these values. For convenience, we number the tree vertices in a breadth-first manner starting at *0*. That is, the root is $v[0]$, the left child of the root is $v[1]$, and the right child is $v[2]$, etc. At each internal vertex v, it assigns the label $b(v[i])=H(b(v[2i+1]), b(v[2i+2]))$ where H is a cryptographic hash function for which it is hard to find two distinct inputs that map to the same value. Finally, the user digitally signs the root value of this tree $b(v[0])$. We also assume that the verification key for the signature is part of a certificate that is signed by the HSP. When the user wishes to use the service, he first sends the root, signature, and certificate to FSP. The FSP verifies these values. At each interval i, the user transmits $b[2i-1]$ and $b[2i]$. The FSP verifies that $b(v[i-1])=H(b(v[2i-1]), b(v[2i]))$, and accepts the token if so. Observe that FSP received $b(v[i-1])$ in an earlier interval. Now, suppose service is terminated after t intervals. The FSP transmits $b(v[2i-1])$, together with the values $b[j]$ of the siblings of all the vertices on the path from $v[2i-1]$ to the root of the tree, to the HSP. The FSP also provides the user's original certificate and signature on tree $b(v[0])$. With these values, the HSP can then compute what should be $b(v[0])$, and can use that to verify the user's signature. If it matches, the HSP bills the user for t intervals of service. More details can be found in [9].

The security proof of the scheme shows that any attempt by the FSP to over-bill the user would result in either forging the underlying digital signature scheme or finding two inputs to the hash function that map to the same value. Since there are signature schemes for which forgery is believed to be infeasible and since there are cryptographic hash functions for which finding colliding inputs is believed to be infeasible, it follows that we can implement this scheme with a high degree of security.

3.2 Swiss-Cheese Authentication

We developed two schemes, LInear multiplex Scheme for Simulcast Authentication (LISSA) and TREe Scheme for Simulcast Authentication (TRESSA), for efficient authentication of media streams that may be modified by a proxy in a content distribution network. Using our schemes, a proxy can intercept a stream digitally signed by a content provider, and modify it dynamically while preserving the ability of the ultimate receiver to verify the content provider's signature and, hence, the authenticity and integrity of the data received. These authentication schemes allow a content provider to encode and sign its entire data stream only once, as opposed to the very expensive processes of either signing each frame or of encoding and signing different versions for each anticipated combination of device, network configuration and channel quality. The proxy, in turn, may be provided as a network service, and be billed

*2 Numbers taken from: http://www.eskimo.com/~weidai/benchmarks.html.

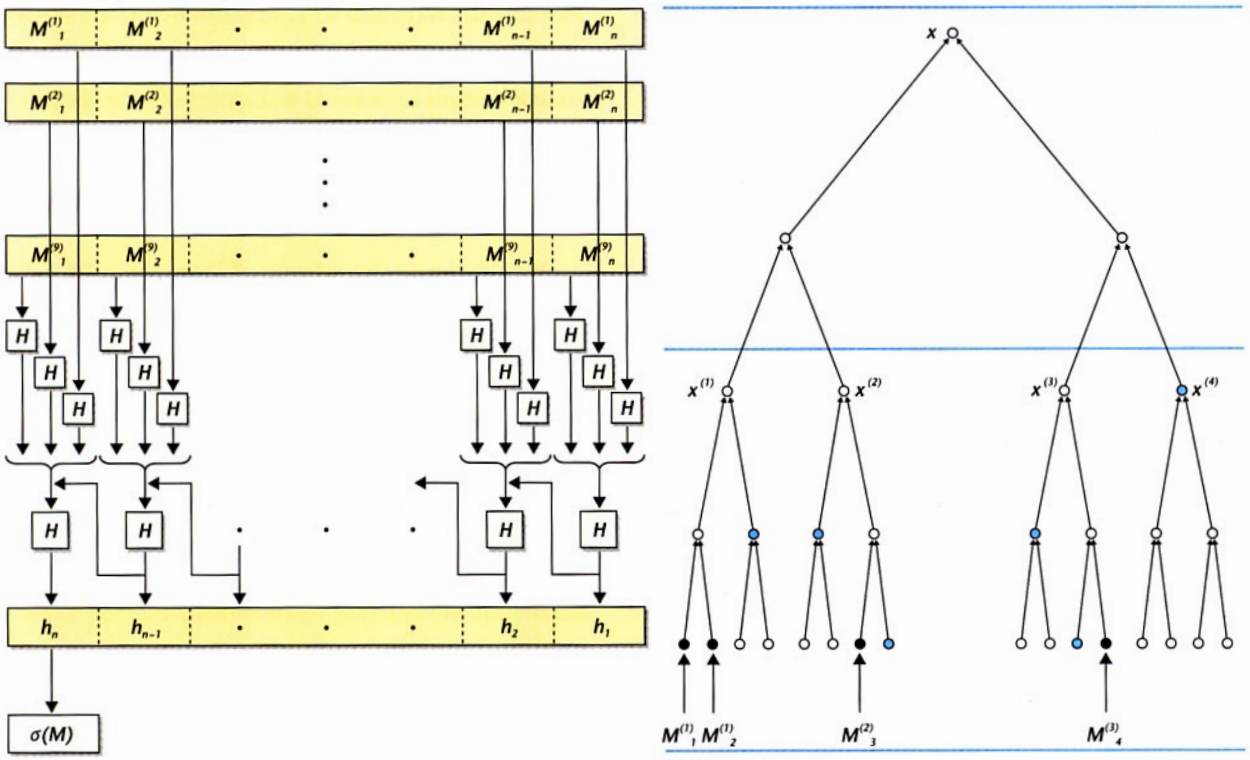
appropriately.

Our schemes are applicable to any setting in which a proxy wishes to remove a portion of data from a stream or other data set without inhibiting the ability of the ultimate recipient to verify the digital signature. For example, one common transcoding technique is multiple file switching, wherein the proxy has several versions of the same stream (e.g., low, medium, and high quality) and chooses which stream to transmit according to the channel conditions. The proxy may want to periodically switch between streams. Another common technique is scalable compression, wherein the multimedia object is broken up into a base layer and several enhancement layers. The base layer alone is sufficient to view the stream whereas the enhancement layers improve the quality. If the network suddenly becomes congested, the proxy may wish to remove enhancement layers to accommodate other Quality of Service (QoS) guarantees. Our techniques can also be used for dynamic advertisement placement wherein the source can include a number of advertisements in a stream, and the intermediary can target particular advertisements according to the preferences of the users it serves. While we have schemes for all these settings, we focus here on multiple-file switching, of which the other scenarios may be viewed as special cases. This scheme is called "Swiss-cheese authentication" because the operations of the proxy leaves 'holes' in the stream received by the recipient so that it resembles Swiss cheese.

At a high level, the schemes work as follows. Both divide a media stream into frames (switchable units of data). Each frame is first separately cryptographically hashed with a first-layer cryptographic hash function. The resulting hashes are then hashed together with a second-layer cryptographic hash function and digitally signed. In the LISSA scheme, the second-layer hash involves a standard iterated chaining construction. In TRESSA, the second-layer hash employs a Merkle tree. These schemes are shown in **Figure 1**.

In Fig.1(a), we have LISSA processing three streams. Each frame is hashed with the first layer hash H, and the resulting hashes are then chained together. That leads to the final hash value h_n, which is digitally signed, with $\sigma(M)$ representing the signature.

In Fig.1(b), we have TRESSA with four streams. Again, each frame is hashed and the values are assigned to the leaf vertices. The interior vertices take on the value associated with the hash of their children. Finally, the root x is signed.



(a) LISSA scheme

(b) TRESSA scheme

Figure 1 The LISSA scheme and the TRESSA scheme

In LISSA if a frame is dropped, we send the first layer hash of that frame instead. In TRESSA, we send the same value—but if any frames are clustered together and form the leaves of an entire subtree, we send the value of the root of that subtree. In Fig.1(b), this value corresponds to the shaded vertices. According to our implementation prototype, the LISSA scheme is anywhere from 3 to 18 times faster than the basic approach of signing each frame. We provide additional details, as well as our comprehensive security analysis [10].

4. Conclusion

In this article we first described three research results related to simplifying PKI: HIBC, certificateless PKC, and aggregate signatures. Next, we described two results related to the secure distribution of content and services. The first was an exceptionally efficient microcredit scheme that uses a new type of hash tree and allows HSPs to securely receive accurate billing information from FSP partners; the second was a stream authentication scheme wherein an intermediate proxy dynamically transcodes a stream without breaking end-to-end security.

This article explained our research efforts in the areas of cryptography and information security. We believe that DoCoMo's all-encompassing 4G-vision presents both numerous opportunities and numerous challenges. Our goal is to develop the necessary security technologies and building blocks to not only help realize DoCoMo's vision, but also to open up even greater business opportunities.

References

[1] S. Micali: "Scalable Certificate Validation and Simplified PKI Management," Proc. of 1st Annual PKI Research Workshop, 2002.
[2] A. Shamir. "Identity-Based Cryptosystems and Signature Schemes," Proc. of Crypto 1984.
[3] D. Boneh and M. Franklin: "Identity-Based Encryption from the Wiel pairing," Proc. of Crypto 2001.
[4] C. Cocks: "An Identity-Based Encryption Scheme Based on Quadratic Residues," Proc of Cryptography and Coding, 2002.
[5] C. Gentry and A. Silverberg: "Hierarchical ID-Based Cryptography," Proc. of Asiacrypt 2002.
[6] C. Gentry: "Certificate-Based Encryption and the Certificate Revocation Problem," Proc. of Eurocrypt 2003.
[7] S. S. AL-Riyami and K. G. Paterson: "Certificateless Public Key Cryptography," Proc. of Asiacrypt 2003.
[8] D. Boneh, C. Gentry, B. Lynn and H. Shacham: "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," Proc. of Eurocrypt 2003.
[9] C. Gentry and Z. Ramzan: "Microcredits for Verifiable Foreign Service Provider Metering," Proc. of Financial Cryptography 2004 (To Appear).
[10] C. Gentry, A. Hevia, R. Jain T. Kawahara and Z. Ramzan: "Swiss-Cheese Authentication for Streaming Media," Technical Report.

Abbreviations

CA: Certificate Authority
CBE: Certificate Based Encryption
CPKE: Certificateless Public-Key Encryption
CRL: Certificate Revocation List
CS: Computer Science
DoS: Denial of Service
FSP: Foreign Service Provider
HIBC: Hierarchical Identity-Based Cryptography
HSP: Home Service Provider
IBC: Identity-Based Cryptography
LISSA: LInear multiplex Scheme for Simulcast Authentication
OCSP: Online Certificate Status Protocol
PKC: Public Key Cryptography
PKG: Private Key Generator
PKI: Public Key Infrastructure
QoS: Quality of Service
TRESSA: TREe Scheme for Simulcast Authentication
TTP: Trusted Third Party