US 20090217041A1

(54) **PROVISIONAL SIGNATURE SCHEMES**

(76) Inventors: **Zulfikar Ramzan**, San Mateo, CA (US); **Craig Gentry**, Mountain View, CA (US); **David Molnar**, Berkeley, CA (US)

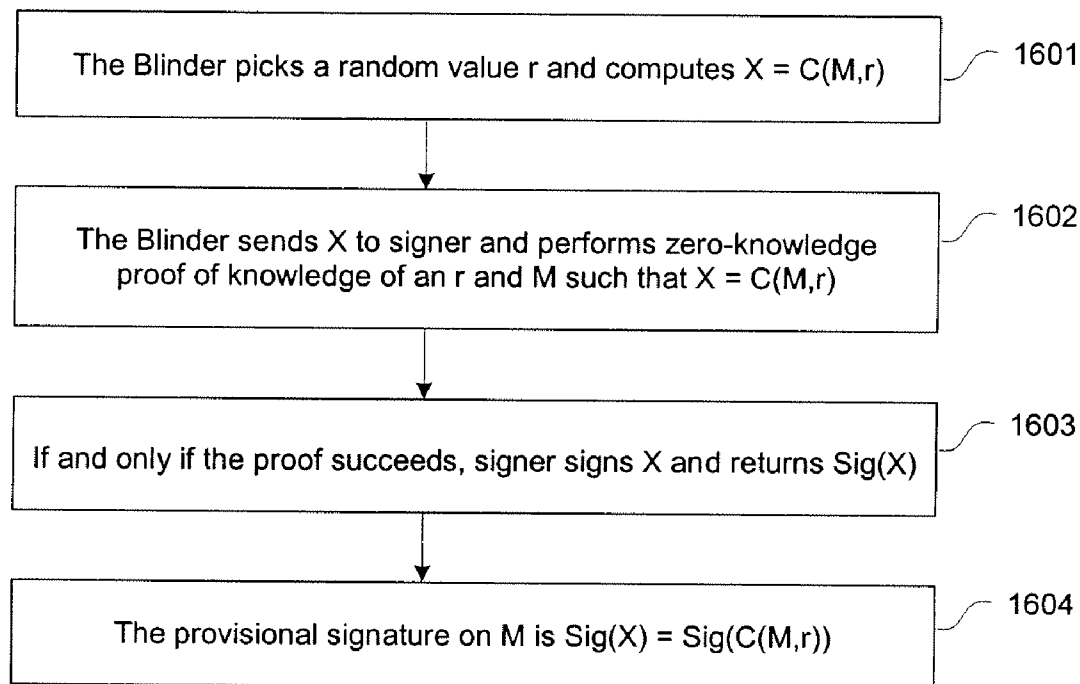Correspondence Address:
BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP
1279 OAKMEAD PARKWAY
SUNNYVALE, CA 94085-4040 (US)

(21) Appl. No.: **12/389,287**

(22) Filed: **Feb. 19, 2009**

(57) **ABSTRACT**

A method and apparatus for implementing portions of a provisional signature scheme are disclosed. In one embodiment, the method comprises creating a provisional signature by performing an operation on a message and completing the provisional signature to create a final signature on the message. Such a scheme may be used for server assisted signature schemes, designated confirmer signature schemes and blind signature schemes.

The Blinder picks a random value r and computes $X = C(M,r)$ — 1601

The Blinder sends X to signer and performs zero-knowledge proof of knowledge of an r and M such that $X = C(M,r)$ — 1602

If and only if the proof succeeds, signer signs X and returns $Sig(X)$ — 1603

The provisional signature on M is $Sig(X) = Sig(C(M,r))$ — 1604

Create a provisional signature by performing
an operation on a message.   101

Complete the provisional signature to create
a final signature on the message.   102

Assign a value to each leaf node of a tree described
by the tree structure data corresponding to the
quantized coefficients   103

Figure 1

M σ Pk

Verifier

Y/N

Figure 2

Processor

301

Memory

(Cryptographic Keys)

302

Communications
network

303

Figure 3

Figure 4

A security parameter k and a number of signatures S is specified. ⟋⟍ 501

A random 160-bit elliptic curve is produced,
together with a generator point g. ⟋⟍ 502

Two seeds s_1 and s_2 are chosen uniformly at random. ⟋⟍ 503

A sequence of values x_i, $1 <= i <= S$, is generated as successive
outputs of a PRG seeded with the random seed s_1. ⟋⟍ 504

A sequence of values c_1, $1 <= i <= S$ is generated as successive
outputs of a PRG seeded with the random seed s_2. ⟋⟍ 505

A sequence of values $h_i =$ $g^{\frac{1}{i}}$, $1 <= i <= S$ is calculated. ⟋⟍ 506

A key pair for a standard secure signature
scheme (SK, PK) is computed. ⟋⟍ 507

$$Sig_i = Sig\left(\left(g^{x_i}h_i^{c_i}, h_i\right)\right)$$

A sequence of S signatures   is computed. ⟋⟍ 508

The seeds s_1 and s_2 are selected by signer.
The values h_i and Sig_i are given to the server.
The public key PK is published as the signer's public key,
together with g and the address of the server. ⟋⟍ 509

Figure 5

The signer generates value x_i by using secret seed s_1 — 601

The signer generates value c_i by using secret seed s_2 — 602

The signer finds value r_i such that $g^{m_i} h_i^{r_i} = g^{x_i} h_i^{c_i}$ — 603

The signer outputs r_i as the provisional signature of m_i — 604

Figure 6

The server is given the index i of the provisional signature r_i, and m_i — 701

The server looks up and returns h_i and $Sig_i = Sig(j_i, h_i)$, where $j_i = g^{x_i} h_i^{c_i}$ — 702

The final signature is $(Sig((j_i, h_i)), h_i, m_i, r_i)$ — 704

Figure 7

The verifier computes $j_i = g^{m_i} h_i^{r_i}$ — 801

The verifier checks whether $V_{PK}(Sig(j_i, h_i)) = 1$ — 802

The verifier accepts if and only if the check passes — 803

Figure 8

The signer creates a key pair (PK_S, SK_S) for a standard secure digital signature scheme — 901

The designated confirmer creates a key pair (PK^{dc}, SK^{dc}) for a semantically secure public-key encryption scheme — 902

Figure 9

The signer creates a commitment $C(M,r)$ to the message M — 1001

The signer creates $S = Sig(C(M,r))$ — 1002

The provisional signatures is $(M, S, E_{pk}(r))$, where E is a semantically secure public-key encryption scheme — 1003

Figure 10

The designated confirmer performs a zero-knowledge proof of knowledge of a value r such that comm = (C(M,r))   ⌐ 1101

Figure 11

Parse the purported signature as $(M, S, E_{PK_{dc}}(r))$ ⌐ 1201

Decrypt $E_{PK_{dc}}(r))$ to recover r   ⌐ 1202

The designated confirmer performs a zero-knowledge proof of knowledge of an r and an Mi such that Ver(S) = 1, $C(M',r) = S$, $D_{ch}(E_{ch}(r)) = r$ and M' = M   ⌐ 1203

Figure 12

The designated confirmer decrypts $E_{PK}(r)$ to obtain r   ⌐ 1301

The designated confirmer outputs (M,r,S) as the final signature on M   ⌐ 1302

Figure 13

The verifier computes comm = C(M,r)   1401

The verifier checks that S is a valid signature on comm under PK_s 1402

The verifier accepts if and only if the check passes   1403

Figure 14

The signer creates a key pair PK_S, SK_S
for a standard secure digital signature scheme   1501

Figure 15

The Blinder picks a random value r and computes X = C(M,r)   1601

The Blinder sends X to signer and performs zero-knowledge
proof of knowledge of an r and M such that X = C(M,r)   1602

If and only if the proof succeeds, signer signs X and returns Sig(X)   1603

The provisional signature on M is Sig(X) = Sig(C(M,r))   1604

Figure 16

The Blinder generates a new random value r' 1701

The final signature on M is $(C(Sig(C(M,r)),r'))$ 1702

Figure 17

The Blinder performs a ZK proof of knowledge to the Verifier of r and r' such that S decommits to the value Sig(C(M,r), Ver(Sig(C(M,r))) = 1, and C(M,r) decommits to M   1801

Figure 18

For a fixed k, find the values x_i in X, where X = {ab : a = $2^{kd'}$, 0 <= d' < d, 1 <= b < 2^k}   1901

The pre-computed values are defined as $(x_i^{g^{x_i}})$ 1902

Figure 19

Compute a Merkle tree on the values ($x_i^i g^{x_i}$) — 2001

The output consists of the values h_i corresponding to the nodes of the Merkle tree, with h_0 as the roof — 2002

Figure 20

Given the values $(x, g^x, h_{i_1}, \ldots, h_{i_n})$, check that the h_i form a valid Merkle authentication path for (x, g^x) — 2101

Figure 21

Given pre-computed values $((y_1, g^{y_1})), \ldots,$ (y_n, g^{y_n}) with authentication paths for each value, and a claimed $\$(x, g^x)\$$, verify pre-computed values — 2201

Check sum of $y_i = x$ — 2202

Check that sum of $g^{y_i} = g^x$ — 2203

Accept if and only if all checks pass — 2204

Figure 22

PROVISIONAL SIGNATURE SCHEMES

PRIORITY

[0001] This is a divisional of application Ser. No. 11/215,550, filed on Aug. 29, 2005, entitled "Provisional Signature Schemes," and assigned to the corporate assignee of the present invention and incorporated herein by reference.

FIELD OF THE INVENTION

[0002] The present invention relates to the field of cryptography; more particularly, the present invention relates to provisional signature schemes, including provisional signatures to construct server assisted digital signatures, designated confirmer signatures, and blind signature schemes.

BACKGROUND OF THE INVENTION

[0003] In the area of general digital signatures, the most common signature schemes are RSA and the U.S. Digital Signature Algorithm over elliptic curves (ECDSA). The RSA algorithm, with appropriate parameters, can be quite fast at verification, but generating signatures is slow. Further, signatures in RSA are at least one kilobyte in size, making them unsuited for SIM cards or for product registration.

[0004] A scheme for "online/offline" digital signatures was proposed by Shamir and Tauman. See A. Shamir & Y. Tauman, "Improved Online-Offline Signature Schemes," CRYPTO 2001. Their scheme made use of chameleon hash functions and introduced the "Hash-Sign-Switch" paradigm that may be used for efficient generation of provisional signatures. They did not, however, consider the application of their scheme to the case of having a server assist in the process.

[0005] In server assisted digital signatures, it is desirable to reduce the computational and communication overhead required for a signature by employing a separate server. This is known as Server Assisted Signatures (SAS). Naturally, one can imagine a number of alternate scenarios wherein efficient digital signatures are desired and some third party is available. The issue of reducing signer communication and computation is of immediate practical interest because it allows for more efficient energy usage and, therefore, longer lifetime for mobile devices. Many previously proposed SAS schemes have been found insecure, while others require the signer to communicate a large amount of data per signature or require the server to store a large amount of state per client.

[0006] An example application for SAS is product registration. A signer may wish to dispense an authorization key for a piece of software or for a newly purchased phone. The verifier comprises the software itself, which is assumed to have connectivity to the server. The authorization key consists of a signature on the software itself plus a serial number. The digital signature is further typed on a piece of paper or a label shipped with the software.

[0007] Another example application for SAS is UIM cards. A UIM card is a smart card containing a processor and a small amount of storage. UIM cards allow the user to maintain a single identity when moving from device to device, such as from one phone to another or from a phone to a PC. In addition, UIM cards are used in FirstPass SSL client authentication, which uses RSA to authenticate a user to a web site. Current UIM cards require special purpose processors to perform RSA digital signatures and may take up to half a second for each signature. Furthermore, an RSA secret key takes one kilobyte of space on the UIM card, and so the number of keys on the card is limited to five.

[0008] Previous solutions to the server-assisted signature problem have several drawbacks. A scheme by Beguin and Quisquater was shown to be insecure by Nguyen and Stem. Therefore, it cannot be considered for practical use. For more information, see P. Nguyen and J. Stem, "The Beguin-Quisquater Server-Aided RSA Protocol from Crypto'95 is not Secure," Asiacrypt 1998 and P. Beguin and J. J. Quisquater, "Fast server-aided RSA signatures secure against active attacks," CRYPTO 1995. A method by Jakobsson and Wetzel appears secure, but is limited to use for only DSA and ECDSA, because signatures are at least 320 bits in size. See M. Jakobsson and S. Wetzel, "Secure Server-Aided Signature Generation," International Workshop on Practice and Theory in Public Key Cryptography, 2001.

[0009] A scheme by Bicacki and Bayal requires the server to store five kilobytes per signer per signature. See Bicacki & Bayal, "Server Assisted Signatures Revisited," RSA Cryptographers' Track 2003. If there were, for example, 80 million signers, each of whom produce 10 signatures per day, this requires storing roughly 3.7 terabytes per day. The scheme of Goyal addresses this problem and requires 480 bits of server storage per signature. See, V. Goyal, "More Efficient Server Assisted Signatures," Cryptography Eprint Archive, 2004. With 80 million signers, 10 signatures per day, this scheme requires roughly 357 gigabytes per day.

[0010] Worse, in both schemes, the amount of data the server must store increases without bound. This is because the data is kept in case the server is accused of cheating by some signer. Therefore, the data must be kept until the server is sure it cannot be accused of cheating, which in practice may be months or years. Assuming a "statute of limitations" period of one year, Goyal's scheme requires more than 127 terabytes of server storage. If any data is missing and a signature is challenged, the server will be unable to prove it acted correctly.

[0011] Another drawback of both the Goyal and the Bicacki-Bayal schemes is that the signer must send a public key for a one-time signature to the server for each message. With the suggested embodiment of Goyal's paper, this requires 26 kilobytes of communication per signature. This large communication makes the product registration application infeasible.

[0012] Another type of signature is a designated confirmer signature. In designated confirmer digital signatures, a signature on a message cannot be verified without the assistance of a special "designated confirmer." The signer selects the designated confirmer when the signature is generated. The designated confirmer can then take a signature and either confirm that the signature is genuine, or disavow a signature that was not actually created by the signer, but the confirmer cannot generate any new signatures. Further, the confirmer can convert a signature into a regular signature that can be verified by anyone.

[0013] An example application of using a designated confirmer is the signing of electronic contracts. A job candidate and a potential employer may negotiate an employment contract without being physically present in the same room. The employer would prefer that the employee not use the contract as a bargaining tool with other prospective employers. Therefore, the employer can sign using a designated confirmer signature and designate a court of law as the confirmer. That way, if a dispute arises, the signature can be verified, but the signature cannot be verified in the meantime by other employ-

ers. After both parties have finalized the contract, the signature can be converted to a regular signature.

[0014] Another example application for use of a designated confirmer is the verification of software patches. A software vendor may wish to restrict software patches only to users who have properly paid for software. One method of accomplishing this restriction is to sign patches with a designated confirmer signature scheme and provide confirmation only to registered users. Unregistered users cannot verify the signature and run the risk of installing compromised software patches.

[0015] Most previous implementations of designated confirmer digital signatures use special-purpose properties of algorithms such as RSA. If these specific algorithms are found insecure, then these schemes are also insecure. Goldwasser and Waisbard showed how to convert several existing signature schemes into designated confirmer signature schemes. See, S. Goldwasser and E. Waisbard, "Transformation of Digital Signature Schemes into Designated Confirmer Signature Schemes," Theory of Cryptography Conference, 2004.

[0016] Another type of signature is a blind signature. In blind digital signatures, the signer signs a "blinded" version X of the message M. The blinded version X is generated with the aid of a blinding factor r. A blinder wishes to obtain a signature on a message M without revealing M to the signer. This is achieved by the blinded asking the signer to sign a message X, which is the "blinded version" of M. After signing, the signature can be "unblinded" using the blinding factor to obtain a signature on M. Without the blinding factor, it is infeasible to link a signature on the blinded message X with a signature on the un-blinded message M. From the signature on X, the blinder can then recover a signature on M. The signature on X as the "provisional signature," and the signature on M as the "final signature."

[0017] An example application of blind signatures is unlinkable electronic cash tokens. Our goal is to enhance user privacy by ensuring not even the bank can track different transactions. The user creates a token for a certain denomination and then blinds the token. The bank signs the blinded token and returns it to the user, who unblinds to obtain the bank's signature on a token. With the token and bank's signature on the token, the user can partake in a financial transaction since a third party can verify the bank's signature. On the other hand, because the bank signed the blinded token, it cannot trace the token back to the user, hence providing anonymity for the user. To avoid cheating users, a cut and choose protocol may be used in which the user generates 100 or more tokens of the same denomination and the bank asks to see 99 of them, chosen randomly, before signing the last token.

SUMMARY OF THE INVENTION

[0018] A method and apparatus for implementing portions of a provisional signature scheme are disclosed. In one embodiment, the method comprises creating a provisional signature by performing an operation on a message and completing the provisional signature to create a final signature on the message. Such a scheme may be used for server assisted signature schemes, designated confirmer signature schemes and blind signature schemes.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] The present invention will be understood more fully from the detailed description given below and from the accompanying drawings of various embodiments of the invention, which, however, should not be taken to limit the invention to the specific embodiments, but are for explanation and understanding only.

[0020] FIG. 1 is a flow diagram of one embodiment of a provisional signature process.

[0021] FIG. 2 is a flow diagram of one embodiment of a verifier process;

[0022] FIG. 3 illustrates of one embodiment of a signer, server, or verifier component.

[0023] FIG. 4 illustrates an exemplary computer system.

[0024] FIG. 5 is a flow diagram of one embodiment of a process for server assisted key generation.

[0025] FIG. 6 is a flow diagram of one embodiment of a process for provisional signing in a server assisted key generation scheme.

[0026] FIG. 7 is flow diagram of one embodiment of a process for completing a provisional signature in a server-assisted signature scheme.

[0027] FIG. 8 is a flow diagram of one embodiment of a process for verifying a final signature in a server-assisted signature scheme.

[0028] FIG. 9 is a flow diagram of one embodiment of a process generating keys for a designated confirmer signatures scheme.

[0029] FIG. 10 is a flow diagram of one embodiment of a process for generating keys for a designated confirmer signature scheme.

[0030] FIG. 11 is a flow diagram of one embodiment of a process for confirmation for a designated confirmer signature scheme.

[0031] FIG. 12 is a flow diagram of one embodiment of a process for disavowal for a designated confirmer signature scheme.

[0032] FIG. 13 is a flow diagram of one embodiment of a process for a designated confirmer signature scheme.

[0033] FIG. 14 is a flow diagram of one embodiment of a process for verification of final signature for a designated confirmer signature scheme.

[0034] FIG. 15 is a flow diagram of one embodiment of a process for a key generation for blind signature scheme.

[0035] FIG. 16 is a flow diagram of one embodiment of a process for provisional generation for a blind signature scheme.

[0036] FIG. 17 is a flow diagram of one embodiment of a process for completion of a blind signature.

[0037] FIG. 18 is a flow diagram of one embodiment of a process for verification of a blind signature.

[0038] FIG. 19 is a flow diagram of one embodiment of a process for pre-computating a chameleon hash function.

[0039] FIG. 20 is a flow diagram of one embodiment of a process for certifying pre-computed values for verification of a chameleon hash function.

[0040] FIG. 21 is a flow diagram of one embodiment of a process for verifying pre-computed values for verification of a chameleon hash function.

[0041] FIG. 22 is a flow diagram of one embodiment of a process for checking a chameleon hash using pre-computed values for verification of the chameleon hash function.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

[0042] Provisional signature schemes are described. Specifically, the signer produces "provisional signatures" that are

converted to "final signatures" by a third party server. These final signatures can then be verified by a verifier. In one embodiment, to compensate to the possibility that the third party server could be compromised or malfunctioning, the server cannot sign documents on its own but only convert provisional signatures created by the signer. In one embodiment, the server performs little computation and stores only a small amount of data per signature, so that the server may scale to handle a large number of signers.

[0043] The provisional signature schemes include server assisted signature schemes, designated confirmer signature schemes, and blind signature schemes. In one embodiment, the server assisted signatures is used in product registration and in reducing the computational load on a device. In one embodiment, the designated confirmer signature schemes are used for fair exchange of digital contracts. In one embodiment, the blind signature schemes are used to create anonymous electronic cash.

[0044] Embodiments of the present invention include schemes for secure server assisted signatures that are efficient with respect to the computation requirements of the signer, server and verifier, as well as the bandwidth requirements of the channels over which these parties communicate. In one embodiment of the present invention, the scheme has a communication complexity of the signer of only 160 bits per signature, which is an order of magnitude improvement over previous SAS schemes.

[0045] In one embodiment, the SAS method described herein for use with UIM cards requires only 128 bits of space for each secret key, and it allows fast signatures without use of special purpose co-processors.

[0046] In the following description, numerous details are set forth to provide a more thorough explanation of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the present invention.

[0047] Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0048] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic com-

puting device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0049] The present invention also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories (RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

[0050] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will appear from the description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the invention as described herein.

[0051] A machine-readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory ("ROM"); random access memory ("RAM"); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

DEFINITIONS AND ASSUMPTIONS

[0052] For purposes herein, a function f(n) is negligible if for any polynomial function q(n), there is a value n_0 such that for all $n > n_0$, it holds that $f(n) < 1/q(n)$. One example of such a negligible function is $f(n) = 1/2^n$.

[0053] If a and b are two integers with $a \leq b$, [a, b] denotes the set of integers between a and b inclusive. That is, $[a, b] = \{c \in Z | a \leq c \leq b\}$.

[0054] If S is a set of elements, and D is a sample-able probability distribution on S, the process of picking an element s from S according to the distribution D is denoted by

$$s \xleftarrow{D} S.$$

[0055] It is known to one of ordinary skill in the art that the security of many cryptographic techniques relies upon making certain computational intractability assumptions. For example, one may try to prove that a cryptosystem is secure so long as it is difficult to decompose a specific number into its prime factors efficiently. The term "computational" is often used to identify this class of cryptographic techniques. In one embodiment, a set of assumptions relevant to proving the

4

security of the embodiments of the present invention described herein are described below.

The Discrete Logarithm Assumption

[0056] The discrete logarithm assumption in a group G states that given a generator g of the group, and given a value $y=g^x$, it is computationally difficult to obtain x. We will be specifically interested in the group of rational points of an elliptic curve. This constitutes a standard mathematical group on which to define the discrete logarithm problem. For such an elliptic curve group of order q, the best known approaches for finding the discrete logarithm require time $\sqrt{q}$.

Chameleon Hash Function

[0057] A chameleon hash function is a function CH(m, r) defined by a public key PK_{ch} and a secret key SK_{ch} generated by a probabilistic polynomial time algorithm $G(1^k)$. Given the public key PK_{ch} it is easy to evaluate CH(m, r). Without the secret key, it is hard to find a tuple (m, m', r, r') such that CH(m, r)=CH(m', r'). With the secret key, on the other hand, it is easy, given m, m', and r, to find an r' such that CH(m, r)=CH(m', r').

[0058] A specific family of chameleon hash functions is defined for a group G of order q in which the discrete logarithm assumption holds as follows. The secret key SK_{ch} is a uniform random value x in Z^*_q, while the public key is the value $h=g^x$. Then defined such that CH(m, r) is CH(m, r)=$g^m h^r$. For a specific group instance, G is the group of points on an appropriately chosen elliptic curve. By appropriate choices of parameters, a chameleon hash with outputs 160 bits in length is obtained. Embodiments of the present invention described herein include the use of this chameleon hash function; however, it will be apparent to anyone with ordinary skill in the art that another chameleon hash function could be used.

Pseudo-Random Generator

[0059] A pseudo-random generator G takes as input a short random seed of s bits and outputs a string of k bits where k>s. The output string is pseudo-random in the sense of being indistinguishable from a random string.

Bit Commitment Scheme

[0060] A bit commitment scheme C(M, r) satisfies the property of being statistically hiding and computationally binding. Statistically hiding means that no adversary, no matter how powerful, can recover M from C(M, r) without knowledge of r except with negligible probability. Computationally binding means that the commitment C(M, r) cannot be opened to a value M' ≠M by any probabilistic polynomial time algorithm.

Overview

[0061] FIG. 1 is a flow diagram of one embodiment of a provisional signature process. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[0062] Referring to FIG. 1, the process begins by processing logic creating a provisional signature by performing an operation on a message (processing block 101). Next, processing logic completes the provisional signature to create a

final signature on the message (processing block 102). Both processing blocks 101 and 102 may be performed using one or 2 secret keys. After the final signature has been completed, processing logic verifies the final signature (processing block 103).

[0063] FIG. 2 is a flow diagram of one embodiment of a verifier process. Referring to FIG. 2, the signature generated through the use of the provisional signature process described herein is verified by inputting into the signature, the message m and a public key into verifier and receiving an indication (e.g., yes/no) of whether the signature is valid.

[0064] The provisional signature process described herein may be applied to server assisted signature schemes, designated confirmer schemes and blind signature schemes. Embodiments of these schemes are described below.

An Example System for Server Assisted Signatures

[0065] In one embodiment, a system for communicating data between a signer, verifier, and server provides server assisted signatures. The signer generates provisional signatures and transmits the generated provisional signatures over a communications network to a verifier. The verifier transmits a provisional signature over a communications network to a server. Subsequently, the verifier receives a final signature from the server and verifies the resulting final signature. The server converts the provisional signature to a final signature and transmits the final signature over a communications network to the verifier.

[0066] Each of the signer, verifier, and server comprise a component having processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software apparatus (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. FIG. 3 illustrates one embodiment of such a component. Referring to FIG. 3, component 300 includes a processor 301, memory 302 and a network interface 303. Processor 301 is coupled to memory 302 and network interface 303.

[0067] The signer includes processor logic 301 with processing logic to receive a message through an external network interface 303 and apply a method to create provisional signatures to thereby obtain a provisional signature on the message.

[0068] The verifier includes processor 301 with processing logic to receive a message and a final signature through an external network interface 303 and to apply a method for verifying signatures to thereby obtain assurance that the message originated with the indicated signer.

[0069] The server includes processor 301 with processing logic to transmit to network 304 the output given by a method for converting a provisional signature received through an external network interface 303 as an input as part of a network request into a final signature.

[0070] FIG. 5 is a flow diagram of one embodiment of a process for a key generation for a server assisted digital signature technique. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, key generation is performed by the signer.

[0071] Referring to FIG. 5, the process begins by processing logic specifying a security parameter k and a number of signatures S (processing block 501). Next, processing logic produces a random 160-bit elliptic curve, together with a

generator point g (processing block **502**). There are standard techniques for selecting a generator. For example, one approach is to pick a random element and see if it happens to be a valid generator. In one embodiment, this curve and generator are used for all entities in the system. For purposes herein, and as a break with convention, a group of points over an elliptic curve is notated as a multiplicative group; it will be apparent to one of ordinary skill in the art how to transfer such notation to the standard additive notation. Note that other elliptic curves could be used (e.g., a 161-bit elliptic curve, as well as a 1024-bit finite field. In general, the scheme could use any algebraic group in which the discrete logarithm is hard, and the generator g should come from that group.

[0072]   After the curve and generator are produced, processing block chooses uniformly two seeds s1 and s2 at random (processing block **503**). Next, processing logic generates a sequence of values x_i, where $1 \leq i \leq S$ as successive outputs of a PRG seeded with the random seed s_1 (processing block **504**) and generates another sequence of values c_i, $1 \leq i \leq S$ as successive outputs of a PRG seeded with the random seed s_2 (processing block **505**). Once the two sequences are generated, processing logic calculates a sequence of values $h_i = g^{xi}$, where $1 \leq i \leq S$ (processing block **506**). Then, processing logic computes a key pair for a standard signature scheme (SK, PK) (processing block **507**). The secret key SK is used to create a sequence of S signatures $Sig_i = Sig((g^x, h_i^{c_i}, h_i))$.

[0073]   Once the calculations have been completed, processing logic sends the seeds s_1 and s_2 to the signer (processing block **508**) and sends the values h_i and Sig_i to the server (processing logic **509**). Also, processing logic publishes the public key PK as the signer's public key, together with g and the address of the server.

[0074]   FIG. **6** is a flow diagram of one embodiment of a process for generating a provisional signature for use in a server assisted signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is a part of the signer.

[0075]   Referring to FIG. **6**, the process begins by processing logic generating the value x_i by using its secret seed s_1 (processing block **601**) and generates the value c_i by using its secret seed s_2 (processing block **602**). Next, processing logic finds an r_i such that that $g^{m_i} h_i^{r_i} = g^x h_i^{c_i}$ (processing block **603**) and outputs r_i as the provisional signature of message m_i (processing block **604**).

[0076]   FIG. **7** is a flow diagram of one embodiment of a process for completing a provisional signature for a server assisted signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the server.

[0077]   Referring to FIG. **7**, the process begins by processing logic receiving the index i of the provisional signature r_i (processing block **701**). Next, processing logic looks up and returns the corresponding values of the sequence of signatures $Sig_i = Sig((g^{x_i} h_i^{c_i}, h_i))$ and h_i (processing block **702**) and outputs the final signature $(Sig((g^{x_i} h_i^{c_i} h_i)), h_i, m_i, r_i)$ (processing block **703**).

[0078]   FIG. **8** is a flow diagram of one embodiment of a process for verifying a final signature for a server assisted

signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the verifier.

[0079]   Referring to FIG. **8**, the process begins by processing logic computing $g^{m_i} h_i^{r_i}$ (processing block **801**) and verifying the signatures by checking $V_{PK}(Sig(g_i h_i)) = 1$ (processing block **802**). Next, processing logic accepts the signature as valid if and only if the check passes (processing block **803**).

[0080]   Alternatively, the server assisted signature scheme may be viewed a server-assisted one-time signature scheme by using the "hash-sign-switch paradigm" to transform any underlying signature scheme secure against existential forgery under chosen message attack. It is assumed that an existing signature scheme (Gen, Sig, Ver) is secure against existential forgery under adaptive chosen message attack and that S signatures in total are to be signed.

[0081]   1. Key Generation: First generate a key pair PK_{sig} and SK_{sig} for the underlying signature scheme. Then, using a PRG with seed s_1, generate a sequence of chameleon hash key pairs (SK_{ch}^i, PK_{ch}^i) for i from 1 to S. Finally, using PRG with seeds s_2 and s_3, generate two sequences of pseudo-random values $v_1, \ldots, v_s$ and $w_1, \ldots, w_s$. The variable c_i is defined such that $c_i = CH_i(w_i, v_i)$—i.e., the chameleon hash of (w_i, v_i) under the chameleon hash key PK_{ch}^i. SK_{prov} consists of the seeds s_1 and s_2, while SK_{comp} consists of the values PK_{ch}^i, $Sig(c_i, PK_{ch}^i)$. The public key PK consists of PK_{sig} and the address of the server.

[0082]   2. ProvSign: On input (M, i) for the next value i, compute SK_{ch}^i using s_1, v_i using s_2, and w_i using s_3. Then compute r_i such that $CH_i(M, r_i) = CH_i(v_i, w_i)$. Return r_i as the provisional signature on M. Mark the value i as used.

[0083]   3. Complete: On input (±, i), return PK_{ch}^i and $Sig(c_i, PK_{ch}^i)$. The final signature is then (M, r_i, PK_{ch}^i, $Sig(c_i, PK_{ch}^i)$).

[0084]   4. Verify: On input (M, r_i, PK_{ch}^i, $Sig(c_i, PK_{ch}^i)$), accept the signature as valid if and only if $Ver(Sig(CH_i(M, r_i), PK_{ch}^i) = 1$

[0085]   Note that the entire "secret key" for the Complete operation, SK_{comp}, can be revealed without enabling an adversary to forge final signatures. Therefore, a server in this server-assisted signature scheme may be aggressively replicated. Furthermore, the server performs no computation, but simply returns static, read-only values P_{ch} and $Sig(c, PK_{ch})$.

[0086]   In one embodiment, the system for communicating data between signer, verifier, and server for performing server assisted digital signatures comprises a client component capable of creating provisional signatures, a server component capable of completing provisional signatures to yield final signatures, and a verifier component capable of verifying final signatures.

[0087]   In one embodiment, each of the signer, verifier and the server of an implementation of a server assisted signature scheme may be a hardware apparatus (e.g., circuitry, dedicated logic, etc.), software apparatus (such as is run on a general purpose computer system or a dedicated machine), or a combination of both, capable of performing processing logic. Each of these components may be implemented as the component shown in FIG. **3**. The server-assisted signer uses the external network interface to receive a request for a provisional signature and its processor, which is coupled to the external network interface and the memory, to create the

6

provisional signature and return the provisional signature via the external network to the requesting party. The server assisted signature verifier component uses its external network interface to receive a final signature. The server assisted signature server component uses its external network interface to receive a provisional signature and its processor, which is coupled to the external network interface and the memory, to transmit to the network the completed final signature for a server assisted signature scheme.

[0088] In one embodiment of a server-assisted signature scheme, the chameleon function $CH(m, r)=g^x h^r$. In such a case, the signor storage need only store the seed s, which is 128 bits in length, and a counter, which is 20 bits in length, to represent the variable "i" used in the description below. Thus, the total signer storage is 148 bits, regardless of the number of signatures. Note that most previous public-key signature schemes, such as RSA, require much larger secret key sizes. With respect to signor computation, the signer evaluates the PRG a constant number of times to obtain x_i, and then performs $O(\log^2 q)$ operations to compute the provisional signature r_i, where q is the order of the group G. The signer need only communicate r_i, which is 160 bits. The server includes storage that, for each signature, stores h_i and Sig_i. In this embodiment, the value h_i is 160 bits, while by using an appropriately short signature scheme, Sig_i can also be reduced to 320 bits or less. The server does not perform any on-line computation. Instead, the server simply retrieves the pair (h_i, Sig_i) and returns it to the verifier. As far as verifier computation is concerned, the verifier must perform one elliptic curve point multiplication, and one ordinary signature verification.

An Example of a Designated Confirmer Scheme

[0089] The process of creating a provisional signature and completing the provisional signature may be used for designated confirmer schemes. An example of such a scheme is given below.

[0090] FIG. 9 is a flow diagram of one embodiment of a process for generating a key for use in a designated confirmer signatures scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of a signer or designated confirmer.

[0091] Referring to FIG. 9, the process begins by processing logic creating a key pair PK_S, SK_S for a standard secure digital signature scheme (processing logic 901) and creating a key pair PK_{dc}, SK_{dc} for a semantically secure public-key encryption scheme (processing block 902). This is done in a manner well-known in the art.

[0092] FIG. 10 is a flow diagram of one embodiment of a process for generating a provisional signature for use with designated confirmer signature schemes. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the signer.

[0093] Referring to FIG. 10, the process begins by processing logic creating a commitment $C(M, r)$ to the message M (processing block 1001) and signing the commitment $S=Sig$ $(C(M, r))$ (processing block 1002). Next, processing logic

outputs the provisional signature $(M, S, E_{PK}(r))$, where E is a semantically secure public-key encryption scheme (processing block 1003).

[0094] FIG. 11 is a flow diagram of one embodiment of a process for confirming a provisional signature in a designated confirmer signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the designated confirmer.

[0095] Referring to FIG. 11, the process begins by processing logic performing a zero-knowledge proof of knowledge of a value r, such that comm=$C(M,r)$ where comm is a variable corresponding to the commitment corresponding to provisional signatures and M is the message that the signer has allegedly signed (processing block 1101).

[0096] FIG. 12 is a flow diagram of one embodiment of a process for disavowing a provisional signature for designated confirmer signatures scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the designated confirmer.

[0097] Referring to FIG. 12, the process begins by parsing the purported signature as $(M,S,E_{PK_{dc}}(r))$ (processing block 1201). Then, processing logic decrypts $(M,S,E_{PK_{dc}}(r))$ to recover r (processing block 1202). Finally, processing logic performs a zero-knowledge proof of knowledge of an r and an M' such that Ver(S)=1, $C(M', r)=S$, $D_{ch}(E_{ch}(r))=r$ and M'≠M (processing block 1203).

[0098] In an alternative embodiment, processing logic sends additional information to the verifier to convince the verifier (in zero-knowledge) that the claims message confirmer sends comm., S, and a zero knowledge proof of knowledge of an r such that comm=$C(M,r)$, where M is the message that the signer has allegedly signed. To verify that M was signed, the verifier checks the zero knowledge proof, and checks that S is a valid signature on comm.

[0099] FIG. 13 is a flow diagram of one embodiment of a process for completing a provisional signature in a designated confirmer signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the designated confirmer.

[0100] Referring to FIG. 13, the process begins by processing logic decrypting $E_{pk}(r)$ to obtain r (processing block 1301). Next, processing logic outputs (M, r, S) as the final signature on M (processing block 1302).

[0101] FIG. 14 is a flow diagram of one embodiment of a process for verifying a final signature in a designated confirmer signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the verifier.

[0102] Referring to FIG. 14, the process begins by processing logic computes the variable comm equal to $C(M, r)$ (processing block 1401) and checks that S is a valid signature under PK_s, which is the public key of the signature scheme

(processing block **1402**). Then, processing logic accepts if and only if the check passes (processing block **1403**).

[0103]   In one embodiment, a system for communicating data between signer, verifier, and server for performing designated confirmer digital signatures includes a client component capable of creating provisional signatures, a server component capable of completing provisional signatures to yield final signatures, and a verifier component capable of verifying final signatures.

[0104]   In one embodiment, each of the signer, verifier and the server in one embodiment of an implementation of a designated confirmer signature scheme may be a hardware apparatus (e.g., circuitry, dedicated logic, etc.), software apparatus (such as is run on a general purpose computer system or a dedicated machine), or a combination of both, capable of performing processing logic. Each of these components may be implemented as the component shown in FIG. **3**. The designated confirmer signer uses the external network interface to receive a request for a provisional signature and its processor, which is coupled to the external network interface and the memory, to create the provisional signature and return the provisional signature via the external network to the requesting party. The designated confirmer signature verifier component uses its external network interface to receive a final signature. The designated confirmer signature server component uses its external network interface to receive a provisional signature and its processor, which is coupled to the external network interface and the memory, to transmit to the network the completed final signature for a designated confirmer signature scheme.

Efficiently Realizing the Transformation

[0105]   Using the transformation described herein, the step that may be the most difficult step to perform efficiently is the disavow protocol and the resulting zero-knowledge proof because it simply relies on the fact that "this designated confirmer signature is invalid" is an NP-statement that can be proven in zero knowledge.

[0106]   Interestingly, the possibility that the ciphertext contained in the designated confirmer signature is not well-formed can be eliminated by making some assumptions about the underlying encryption scheme. For example, that a cryptosystem whose outputs is ciphertext-dense if all but a negligible fraction of bit-strings are valid ciphertexts. If the disavow protocol is run with a ciphertext-dense cryptosystem, the bitstring is an invalid ciphertext does not need to be proved.

[0107]   In one embodiment, a ciphertext-dense public-key encryption scheme is constructed from any tradpdoor permutation family as follows. Let the public key be $f:\{0,1\}^k \to \{0,1\}^k$ and the private key be the inverse f^1. To encrypt a single bit b, pick $x,r \leftarrow_R \{0,1\}^k$. The ciphertext is then $(f(x), r, GL(x, r) \oplus b)$, where GL is the Goldreich-Levin predicate. It is easy to see that an adversary breaking the semantic security of the scheme is a predictor for the Goldreich-Levin predicate, and so would contradict the one-wayness of f. Further, the symbol "," means concatenation, every string of 2k+1 bits is a valid ciphertext. Alternatively, an encryption scheme can be used that provides randomness recovery. That is, given an encryption $E_{pk}(m)$ which used randomness r, the decryption includes the randomness r as well as m.

Efficient Instantiations Based on the N-th Residuousity Assumption

[0108]   Essentially, Paillier encryption, which is well-known in the art, works as follows. The recipient chooses a composite modulus N, whose factorization it keeps secret. The recipient also publishes a number $g \epsilon Z/N^2 Z$ that generates a "sufficiently large" group modulo N^2—e.g., a group of order $N\Phi(N)/2$. To encrypt m satisfying $0 \leqq m \leqq N$, the sender chooses a random $r \epsilon Z/N^2 Z$ and sets the ciphertext $c = r^N g^m$ (mod N^2). To decrypt, the recipient essentially computes $m = \log_g c$(mod N). Paillier encryption is semantically secure assuming the decisional N-th residuosity problem is hard.

[0109]   In one embodiment, an efficient designated confirm signature scheme is constructed using any underlying signature scheme, in conjunction with semantically secure Paillier encryption, as follows.

[0110]   1) Key Generation: The signer creates a key pair (PK_S; SK_S) for any standard secure digital signature scheme. The designated confirmer generates a Paillier modulus N and a suitable generator g modulo N^2. It also generates a certificate proving that N has the correct form. Finally, the designated confirmer may also provide the description of a second group G_2 and a generator $g_2 \epsilon G_2$ that has order N.

[0111]   2) ProvSign(M):

[0112]   a) The signer creates a commitment to the message m by generating a random $h_2 \epsilon G_2$ and a random $r \epsilon Z/NZ$, computing $c = g^r{}_2 h^m{}_2 \epsilon G_2$.

[0113]   b) The signer creates $S = Sig(c, h_2)$.

[0114]   c) The signer creates a Paillier encryption of $r \epsilon Z/NZ$ by generating a random $a \epsilon Z/NZ$ setting $r' = r + aN$, and setting $E_N(r) = g^{r'}$(mod N^2).

[0115]   d) The provisional signature is (m, S, c, h_2, $E_N(r)$).

[0116]   3) Confirm by Signer: The signer proves that its designated confirmer signature is correctly constructed by providing a zero-knowledge proof of knowledge of an r and an a such that $E_N(r) = g^r (g^N)^a$(mod N^2) and $c/h_2{}^m = g_2{}^r$. This can be performed using standard techniques. Since the designated confirmer can recover r'(mod N) through Paillier decryption, and since r'(mod N) completely reveals $\log_{g2}(c/h_2{}^m)$ (since G_2 has order N), the verifier is convinced by this zero knowledge proof of knowledge that the designated confirmer can "extract" a conventional signature (m, S, h_2, r) from the designated confirmer signature. Notice that the proof of knowledge can be very efficiently implemented.

[0117]   4) Confirm by Designated Confirmer: To confirm, the designated confirm simply provides the provisional signature and a zero knowledge proof of knowledge of $r = \log_{g2}(c/h_2{}^m)$. It can easily recover r from the Paillier ciphertext.

[0118]   5) Disavow: If the designated confirmer signature is badly formed, either S is not a valid signature on (c, h_2) (which is easily verifiable), or that $c/h_2{}^m \neq g_2{}^{D_N(E_N(r))}$. In other words, if we set $c' = c/h_2{}^m$, it must be the case that $\log_{g2} c' \neq \log(E_N(r))$(mod N). To prove that this inequality holds, the designated confirmer first recovers $d = \log_g(E_N(r))$(mod N) using Paillier decryption. If $x = \Phi(N)$ and $y = d\Phi(N)$, then $E_N(r)^x = g^y$ (mod N^2), but $c'^x \neq g_2{}^y$. The designated confirmer can provide a zero knowledge proof of knowledge of these x and y using fairly standard techniques. In particular, one can construct the usual three-round zero knowledge proof by 1) having the confirmer choose values $u, v \epsilon Z/NZ$ and sending $(A, B) = (E_N(r)^u g^v, c'^u g_2{}^v g^v$ to the verifier, 2) having the verifier randomly choose a bit $b \epsilon \{0, 1\}$, 3) having the confirmer send back values $u', v' \epsilon Z/NZ$ such that $E_N(r)^{u'} g^{v'} = c'^u g_2{}^v g_2{}^{v'} = B$ if b=0 or such that $E_N(r)^{u'} g^{v'} = A$ and $c'^{u'} g_2{}^{v'} \neq B$ if b=1. In the last step, the confirmer can generate such (u',v') by generating a random $0 \neq k \epsilon Z/NZ$ and setting u'=u+bkx(mod N) and v'=v−bky(mod N).

8

[0119] The designated confirmer, since it decrypt the value of r, converts the designated confirmer signature into an "ordinary" signature that can be verified by anyone; this ordinary signature consists of (m, S, h_2, r), and a verifier checks that S is a valid signature on (c, h_2) for $c=g_2'h_2''^m$. However, to prove the confirmer's security—i.e., to prove that malicious adversaries that interact with the designated confirmer will be unable to eventually usurp the role of the confirmer and gain the ability to convert designated confirmer signatures into signatures verifiable by everyone—a semantically secure version of Paillier encryption does not seem to be sufficient.

[0120] However, it is a relatively simple matter to replace the semantically secure version above with an IND-CCA2 secure version of Paillier encryption described by Camenisch and Shoup. The zero knowledge proofs are essentially the same. The main difference is that, since the encryption scheme is IND-CCA2 secure, the confirmer can securely reveal the decryption of ciphertexts chosen by malicious adversaries, and thus can securely extract an ordinary signature from a designated confirmer signature as described above.

An Example of a Blind Signature Scheme

[0121] FIG. 15 is a flow diagram of one embodiment of a process for generating a key for a blind signatures scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the signer.

[0122] Referring to FIG. 15, the process begins by processing logic creating a key pair PK_S, SK_S for a standard secure digital signature scheme (processing block 1501).

[0123] FIG. 16 is a flow diagram of one embodiment of a process for generating a provisional signature for a blind signature scheme. The process is performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the blinder or signer.

[0124] Referring to FIG. 16, the process begins by processing logic picking a random value r and computes $X=C(M, r)$ (processing block 1601). Next, processing logic sends X to the signer and performs a zero-knowledge proof of knowledge of an r and M such that $X=C(M,r)$ (processing block 1602). If and only if the proof succeeds, then processing logic signs X and returns Sig(X) (processing block 1603) and outputs the provisional signature on M as $Sig(X)=Sig(C(M, r))$ (processing block 1604).

[0125] FIG. 17 is a flow diagram of one embodiment of a process for completing provisional signature for a blind signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the blinder.

[0126] Referring to FIG. 17, the process begins by processing logic generating a new random value r' (processing block 1701). Next, processing block begins by processing logic outputting the final signature on M as (C(Sig(C(M, r)), r')) (processing block 1702).

[0127] FIG. 18 is a flow diagram of one embodiment of a process for verifying a final signature for a blind signature scheme. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the blinder or a signer.

[0128] Referring to FIG. 18, the process begins by processing logic performing a ZK proof of knowledge to the Verifier of r and r' such that S decommits to the value Sig(C(M, r)), Ver(Sig(C(M, r)))=1, and C(M, r) decommits to M (processing block 1801).

[0129] The above interactive proof can be rendered non-interactive using random oracles via the "Fiat-Shamir heuristic."

[0130] FIG. 19 is a flow diagram of one embodiment of a process for generating pre-computed values for verification of a chameleon hash function. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of the blinder or signer.

[0131] The problem may be set forth as given (G, g, y, x), where G is a group of order q, where g, $y \in G$, and $x \in [1, q]$, prove that $g^x=y$ in G. Suppose, for convenience, that $d \log q(c_{max}+1) e=kd$ for some integer d, where c_{max} is the maximum value the challenge c can take, and where k is the integer parameter mentioned above.

[0132] Referring to FIG. 19, the process begins by processing logic, for a fixed k, finding the values $x_i \in X$, where $X=\{ab: a=2^{kd'}, 0 \leq d' \leq d, 1 \leq b \leq 2^k\}$ (processing block 1901). Thereafter, processing logic defines pre-computed values are defined as (x_i, g^{x_i}) (processing block 1902).

[0133] FIG. 20 is a flow diagram of one embodiment of a process for certifying pre-computed values for verification of a chameleon hash function. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[0134] Referring to FIG. 20, the process begins by processing logic computing a Merkle tree on the values (x_i, g^x_i) (processing block 2001). Next, processing logic outputs the output consists of the values h_i corresponding to the nodes of the Merkle tree, with ho as the root (processing block 2002).

[0135] FIG. 21 is a flow diagram of one embodiment of a process for verifying pre-computed values for verification of a chameleon hash function. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is the blinder or signer.

[0136] Referring to FIG. 21, processing logic gives the values $(x, g^x, h_{i_1}, \ldots, h_{i_n})$, and checks that the h_i form a valid authentication path for (x, g^x) (processing block 2101).

[0137] FIG. 22 is a flow diagram of one embodiment of a process for checking a chameleon hash using pre-computed values for verification of a chameleon hash function. The process may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, etc.), software

(such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[0138] Referring to FIG. 22, the process begins by processing logic giving pre-computed values $((y1, g^{y1}), \ldots, (y_n, g^{yn}))$ with authentication paths for each value, and a claimed (x, g^x), and verifies pre-computed values (processing block 2201). Next, processing logic checks $\Sigma y_i = x$ (processing block 2202). Thereafter, processing logic checks that $\Sigma g^{yi} = g^x$ (processing block 2203). Processing logic accepts the chameleon hash function if and only if all checks pass (processing block 2204).

[0139] In one embodiment, the system for communicating data between signer, verifier, and server includes a client component capable of creating provisional signatures, a server component capable of completing provisional signatures to yield final signatures, and a verifier component capable of verifying final signatures.

[0140] In one embodiment, each of the signer, verifier and the server of an implementation of a blind signature scheme may be a hardware apparatus (e.g., circuitry, dedicated logic, etc.), software apparatus (such as is run on a general purpose computer system or a dedicated machine), or a combination of both, capable of performing processing logic. Each of these components may be implemented as the component shown in FIG. 3. The designated confirmer signer uses the external network interface to receive a request for a provisional signature and its processor, which is coupled to the external network interface and the memory, to create the provisional signature and return the provisional signature via the external network to the requesting party. The blind signature verifier component uses its external network interface to receive a final signature. The blind signature server component uses its external network interface to receive a provisional signature and its processor, which is coupled to the external network interface and the memory, to transmit to the network the completed final signature for a blind signature scheme.

An Exemplary Computer System

[0141] FIG. 4 is a block diagram of an exemplary computer system that may perform one or more of the operations described herein. Referring to FIG. 4, the computer system may comprise an exemplary client or server computer system. The computer system comprises a communication mechanism or bus for communicating information, and a processor coupled with a bus for processing information. The processor includes a microprocessor, but is not limited to a microprocessor, such as, for example, Pentium, PowerPC, Alpha, etc.

[0142] The system further comprises a random access memory (RAM), or other dynamic storage device (referred to as main memory) coupled to the bus for storing information and instructions to be executed by the processor. Main memory also may be used for storing temporary variables or other intermediate information during execution of instructions by the processor.

[0143] The computer system also comprises a read only memory (ROM) and/or other static storage device coupled to the bus for storing static information and instructions for the processor, and a data storage device, such as a magnetic disk or optical disk and its corresponding disk drive. The data storage device is coupled to the bus for storing information and instructions.

[0144] The computer system may further be coupled to a display device, such as a cathode ray tube (CRT) or liquid crystal display (LCD), coupled to the bus for displaying information to a computer user. An alphanumeric input device, including alphanumeric and other keys, may also be coupled to the bus for communicating information and command selections to the processor. An additional user input device is cursor control, such as a mouse, trackball, trackpad, stylus, or cursor direction keys, coupled to the bus for communicating direction information and command selections to the processor, and for controlling cursor movement on the display.

[0145] Another device that may be coupled to the bus is a hard copy device, which may be used for printing instructions, data, or other information on a medium such as paper, film, or similar types of media. Furthermore, a sound recording and playback device, such as a speaker and/or microphone may optionally be coupled to the bus for audio interfacing with the computer system. Another device that may be coupled to the bus is a wired/wireless communication capability to communication to a phone or handheld palm device.

[0146] Note that any or all of the components of the system and associated hardware may be used in the present invention. However, it can be appreciated that other configurations of the computer system may include some or all of the devices.

[0147] Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and described by way of illustration is in no way intended to be considered limiting.

[0148] Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and described by way of illustration is in no way intended to be considered limiting. Therefore, references to details of various embodiments are not intended to limit the scope of the claims which in themselves recite only those features regarded as essential to the invention.

We claim:

1. A method comprising:

creating a provisional signature by performing an operation on a message, wherein creating a provisional signature by performing an operation on a message comprises

selecting a random value r,

computing a value X equal to the commitment C(M, r), where M is the message,

sending the value X to the signer,

performing a zero-knowledge proof of knowledge of the random value r and the message M such that value X equals to the commitment C(M,r),

signing the value X and returning Sig(x) only if the proof succeeds, and

outputting the provisional signature on M as Sig(X)=Sig(C(M, r)); and

completing the provisional signature to create a final signature on the message.

2. The method defined in claim 1 wherein completing the provisional signature to create a final signature on the message comprises:

generating a second random value r'; and

outputting the final signature on the message M as (C(Sig(C(M, r)), r')).

3. A method comprising:

generating a provisional signature by

selecting a random value r,

computing a value X equal to the commitment C(M, r), where M is the message,

sending the value X to the signer,

performing a zero-knowledge proof of knowledge of the random value r and the message M such that value X equals to the commitment C(M,r),

signing the value X and returning Sig(x) only if the proof succeeds, and

outputting the provisional signature on M as Sig(X)=Sig (C(M, r)); and

transmitting the provisional signature to a verifier via a network.

4. An apparatus comprising:

a processor to generate a provisional signature by

selecting a random value r,

computing a value X equal to the commitment C(M, r), where M is the message,

sending the value X to the signer,

performing a zero-knowledge proof of knowledge of the random value r and the message M such that value X equals to the commitment C(M,r),

signing the value X and returning Sig(x) only if the proof succeeds, and

outputting the provisional signature on M as Sig(X)=Sig (C(M, r)); and

a network interface coupled to the processor to transmit the provisional signature to a verifier via a network.

5. A method comprising:

receiving a provisional signature;

converting a provisional signature into a final signature by

generating a second random value r', and

outputting the final signature on the message M as (C(Sig(C(M, r)), r')); and

sending the final signature to a network location.

6. An apparatus comprising:

a network interface to receive a provisional signature; and

a processor to convert the provisional signature into a final signature by

generating a second random value r', and

outputting the final signature on the message M as (C(Sig(C(M, r)), r')), wherein the network interface sends the final signature to a network location.

*   *   *   *   *