- 9 MAR 2007

SYSTEM AND METHOD FOR GUARANTEEING SOFTWARE INTEGRITY

Field of the Invention

The present invention relates to software tamper resistance, and more specifically to authenticating and validating digital messages including software updates in the field.

Background of the Invention

As digital storage technology and computer networks have advanced, efforts to ensure that digital content is made available for use only by authorised recipients have also progressed. One approach for providing security for digital content information is to distribute the information in encrypted form, and then to distribute necessary decryption information in the form of keys to only legitimate users. Unfortunately, unscrupulous legitimate users can share distributed decryption keys with unauthorised recipients, so there has been an increasing trend toward preventing anonymous sharing by requiring the recipient hardware to identify itself to the distributor of secured digital information as belonging to a particular user. The distributor may be the original vendor of secured digital information, or another party that handles the various security tasks (such as computing and communication) for the vendor.

For example, U.S. Pat. No. 4,658,093 to Hellman discloses a system in which a manufacturer of "base units" (specific hardware instances of user devices that perform computations) assigns a random key to be stored by each particular base unit. When a user wants to use a software package, the user's base unit generates a random number and communicates it to the software manufacturer. The manufacturer generates an authenticator response that is a cryptographic function of the particular base unit's key, the requested software, the number of authorised times the software may be used, and the random number generated by the base unit. The manufacturer then electronically delivers the authenticator response to the user's base unit, which uses the same cryptographic function to generate a check value. (The RSA cryptographic function is used by Hellman; it is described in U.S. Pat. No. 4,405,829 to Rivest et al., which is hereby incorporated by reference.) If the check value and the authenticator response match, the base unit accepts the authenticator response as valid and accordingly increments the number of times that delivered software may be used. The base unit verifies the message from

the manufacturer using a digital signature and a hash of the manufacturer's message.

Digital signatures are known in the art and generate a single-bit yes/no answer to the question "Is this message authentic?". A hash is generally the output of a mathematical function that maps values from a large domain into a smaller range, is one-way in that it is computationally infeasible to find any input which maps to any pre-specified output, and is collision-free in that it is computationally infeasible to find any two distinct inputs which map to the same output. Such hashing functions are well known in the art. Unfortunately, the bidirectional communication that the Hellman system requires is not always available due to the distribution method employed or practical due to the sheer number of base units in the field. Also, the Hellman system requires an authorisation and billing unit to maintain a memory of serial numbers and secret keys used to determine a base unit's secret key from knowledge of the base unit's public serial number.

U.S. Pat. No. 6,105,137 to Graunke et al. describes a similar system for authenticating and verifying the integrity of software modules. U.S. Pat. No. 6,138,236 to Mirov et al. extends this general approach to authenticating firmware programmed in a boot PROM and then using that trusted program code to authenticate a subsequent set of program code. The Mirov et al. system appends a digital signature to a self-extracting executable distribution file, and the distributed software is decrypted using a published public RSA decryption key. A comparison of decrypted hash values deems the self-extracting executable distribution file secure and free from accidental or intentional corruption if successful, or rejects and deletes the file if the comparison fails.

U.S. Pat. No. 6,341,373 to Shaw describes another secure data upgrading method that enables only selected portions of program code to be replaced. Shaw also requires the client device to transmit identification information regarding itself to a remote server before receiving updates from the server.

U.S. Patent 5,343,527 to Moore, teaches a method for providing a reuser of a software component from a reuse library with an indication of whether the software component is authentic and valid, or whether it has been tampered with by some unauthorised entity.

Tamper resistant software is becoming increasingly important because movies, music, text, applications, and databases are now being distributed in digital form with copy protection features. Software pirates might attempt to defeat these copy protection features simply by patching the software used in the player hardware; that is, by presenting a bogus software update to the player such that the player then makes all content accessible whether properly authorised to do so or not. Most companies in the industry rely on digital signatures to check the authenticity of a piece of software. This is not a foolproof approach, however, as the check can be disabled by patching a single instruction in player software.

Digital signets present a better solution to this problem than digital signatures. Digital signets are as difficult to forge as digital signatures, but instead of giving a single yes/no output like a digital signature, they produce an arbitrary sequence of bits **K** that is correct if and only if the hash of the received message is properly related to the signet.

US patents 5,978,482 and 6,038,316 to Dwork et al describe digital signet based systems for protecting digital information where the logic behind extricating decryption keys for accessing the protected information is openly known and operates on an authorisation number generated in response to a user number. The user number uniquely identifies and is valuable to the user, so that the user would be unwilling to disclose it to public view. User numbers could include credit card numbers, phone numbers, central processing unit ID numbers, or other numbers having personal sensitivity to the user. Thus, the user is reluctant to share keys or decrypted content with others for fear that the user number would be divulged and that the misbehaving user would be easily identified.

The hash value of a software program has proven to be a particularly good "user number". Modifications to a software program, such as those made by hackers trying to defeat a content protection scheme, cause its computed hash value to change. Therefore, content protection can be improved when the decryption keys used in a content protection scheme are successfully extricated and used only if the software program is provably intact and unmodified.

This is the typical prior art signet calculation: $K = g_1^h \; g_2^a \bmod M$ where **K** is an output sequence of bits, g_1 and g_2 are public numbers stored with the transmitted digital message itself, **h** is the hash of the message, and **a** is the digital signet. **M** is the public modulus under which

this calculation is performed; in other words, **K** is the remainder after dividing the product g_1^b g_2^a by **M**. **M** is usually a prime number, but does not have to be. The output **K** is the basis for comparison used to guarantee the authenticity and integrity of the message, which may comprise a software update.

The document "Keying Hash Functions for Message Authentication" (Bellare M. Et al, International Associateion For Cryptologic Research, Advances In Cryptology - Crypto 96. 16th Annual Cryptology Conference. Sanda Barbara, August 18-22., 1996, XP000626584, ISBN 3-540-61512-1) discloses a method for verifying the integrity and authenticity of information communicated over an insecure channel between two parties. The method described therein requires the use of a secret key known as a Message Authentication Code shared between the parties.

While the prior art in this field describes worthy accomplishments, there exists a need for further improvements to address unsolved needs. For example, how can the value of **K**, which determines if access to protected information should be allowed, be shielded from attack by those who seek to pirate it and the information it protects? If no verifying transmissions from individual recipients are feasible, how can the software being executed by the recipients be legitimately updated in the field? Any modification to the software running on a user device will generally cause its hash to change, and the subsequently computed **K** value will no longer be correct. Replacing user hardware is generally infeasible, and transmission of new device keys to potentially millions of users also presents readily apparent problems.

Summary of the Invention

The present invention accordingly provides, in a first aspect, a method for guaranteeing message integrity, comprising: transmitting a message together with an encrypted integrity value $\mathbf{h}_0$, said encrypted integrity value $\mathbf{h}_0$ being encrypted with a key based on the hash of said message; decrypting said encrypted integrity value $\mathbf{h}_0$; using said integrity value $\mathbf{h}_0$ together with stored integrity values to perform an integrity calculation; and using the result of said integrity calculation for further processing.

The present invention accordingly provides, in a second aspect, a system for guaranteeing message integrity, comprising: means for transmitting a message together with an encrypted integrity value $\mathbf{h}_0$, said

encrypted integrity value h_0 being encrypted with a key based on the hash of said message; means for decrypting said encrypted integrity value h_0; means for using said integrity value h_0 together with stored integrity values to perform an integrity calculation; and means for using the result of said integrity calculation for further processing.

The present invention accordingly provides, in a third aspect, a computer program product directly loadable into the internal memory of a digital computer, comprising software code portions for performing, when said product is run on a computer, a method for guaranteeing message integrity, comprising: transmitting a message together with an encrypted integrity value h_0, said encrypted integrity value h_0 being encrypted with a key based on the hash of said message; decrypting said encrypted integrity value h_0; using said integrity value h_0 together with stored integrity values to perform an integrity calculation; and using the result of said integrity calculation for further processing.

Thus individual user devices are able to guarantee the authenticity and integrity of digital messages sent by a distribution centre without transmissions from individual user devices to the distribution centre. This is particularly useful for content protection and digital rights management purposes, as in the broadcast distribution of encrypted messages intended to be used only by a set of authorised recipients.

Brief Description of the Drawings

A preferred embodiment of the present invention will now be described by way of example only with reference to the accompanying drawings, in which:

FIG. 1 is a diagram of the new message generation method according to a preferred embodiment of the present invention; and

FIG. 2 is a diagram of the new message authentication and validation method according to a preferred embodiment of the present invention.

Detailed Description of the Preferred Embodiment

The preferred embodiment enables individual user devices that perform computations to guarantee the authenticity and integrity of digital messages sent by a distribution centre using a combination of hardware and software. The preferred embodiment requires no transmission of data from individual user devices to the distribution centre, particularly transmissions including any unique device-identifying information or cryptographic keys. This is particularly useful for content protection and digital rights management purposes, as in the delivery of content protection software that only allows correct extrication of an integrity value **K'** if the software is delivered intact.

The software relies on preferably unique and static hardware-based values to determine whether recipients are authorised to access protected content, but executes only if cryptographically authenticated and validated. The messages may include software upgrades and portions of software programs that become complete when combined with cryptographically determined integrity values. Alternately, the integrity values **K'** themselves may comprise portions of software programs or cryptographic keys.

Individual user devices store in tamper-resistant hardware at least two preferably unique secret numbers g_1 and g_2. The tamper-resistant hardware can be included in, for example, multimedia cards as are widely used in personal computers. Multimedia cards are manufactured by Creative Labs (R), among others (for example, see www.creative.com for a description of the SoundBlaster (R) series of cards), and are well known in the art. Alternately, the individual user devices can comprise a completely secure computing system. It is desirable to have a hybrid hardware/software approach to the problem, so that unique combinations of values of g_1 and g_2 are stored in individual user devices like PC audio cards, but where the same software, such as a common audio card driver program, can be executed by many different cards. The secret values of g_1 and g_2 effectively serve as device keys that preferably uniquely identify a given user device hardware instance.

The distribution centre creates a digital signet **a** and a public modulus **M** that are then combined with the device keys g_1 and g_2 and an original message hash h_0 by the user device to compute an integrity value **K**. Any hashing algorithm may be employed, including those that perform various obfuscation functions. Integrity value **K** is preferably unique to

each individual user device, due to the uniqueness of **a**, g_1 and g_2, and is computed by the user devices as $K = g_1^{h0} \; g_2^{a} \mod M$.

Individual user devices receive and process a subsequent message as follows: the user device calculates a message hash value **h** and identifies the values **s** and **pq** (to be described below) that are sent with the message. The user device calculates a new integrity value **K'** using the new values **h**, **s**, and **pq** and the existing values g_1, g_2, **a**, and **M** as follows: $K' = g_1^{x} \; g_2^{a} \mod M$, where $x = s^h \mod pq$. Prime numbers **p** and **q** are selected as described below. Neither **p** nor **q** are transmitted separately. If the message hash value **h** is correct, then $x = h_0$, and the calculated integrity value **K'** equals the predetermined integrity value **K**. **s** is therefore effectively an encryption of the original message value h_0, with the encryption based on new message hash value **h** and **pq**. Since the user device needs **K** for further correct operation (for example, **K** might contain device keys), the message will be properly processed if and only if its source and content are correct.

The correct hash value h_1 of the new message is designed to be the distribution centre's public RSA key, which can be ensured if there is a **z** such that $h_1 z = 1 \mod F(pq)$, where F is Euler's function, and in this case $F(pq)=(p-1)(q-1)$. **z** then becomes the distribution centre's private RSA key for the new message. For this formula to be satisfied, the greatest common denominator (gcd) of h_1 and $F(pq)$ must be 1, i.e. they must have no common factors. Both (p-1) and (q-1) are even, so if h_1 is even, the relationship can never be satisfied and no **z** exists. Therefore, steps must be taken to ensure that h_1 is odd. For example, inconsequential changes to the new message can be made until the hash value of the new message becomes odd, or a convention can be adopted such that one is always either added to or subtracted from the hash value if it is even. In an alternative example, the low order bit of the hash can be OR'd. In yet another alternative, one could assign $h_1 = 2h_1+1$ to sure that h_1 is forced to be odd. Other techniques for ensuring h_1 is odd will be familiar to persons skilled in the art. If h_1 is odd, it is likely that the greatest common denominator is 1, but testing is required to guarantee this condition. If the greatest common denominator is not 1, the distribution centre simply picks another set of primes **p** and **q** and tries another value of modulus **pq**. It is a common practice in RSA encryption to pick primes as follows: pick a random prime **p'**, and test if 2p'+1 is also prime; if so, set p=2p'+1. Then, in the preferred embodiment, do the same

for **q**. If this practice is followed, the chance that an odd h_1 yields a greatest common denominator greater than 1 is vanishingly small.

Referring now to FIG. 1, a diagram of the new message generation method according to a preferred embodiment is shown. The distribution centre (DC) prepares a new message for distribution to at least one user device (UD). For example, the means for distribution might include downloading over a computer network such as the Internet, satellite and cable television transmission, and physical distribution of computer-readable media such as diskettes, CD-ROMs, and DVD-ROMs. Other means for distribution will be familiar to the person skilled in the art. In step 102, the distribution centre computes a hash h_1 of the new message, and forces the hash to be an odd number by performing any one of the conventional steps described above. Next, the distribution centre selects two prime numbers **p** and **q** in step 104 and computes F(**pq**) and the product **pq**. In step 106, the distribution centre determines if hash h_1 and F(**pq**) have a greatest common denominator of 1, i.e. share no common factors. This condition determines if the relationship $h_1 z = 1 \bmod F(pq)$ is obeyed, so that the hash value h_1 of the new message is the distribution centre's public RSA key. If the condition is not met, the distribution centre selects new values for **p** and **q** in step 108 and then returns to step 104 to compute F(**pq**) and the modulus **pq**. If the condition is met, then the distribution centre proceeds to calculate **z** in step 110 such that $zh_1 = 1 \bmod F(pq)$. This is done with the well known mathematical procedure called the Generalised Euclid Algorithm. The distribution centre proceeds in step 112 to compute $s = h_0^z \bmod pq$, where h_0 is the original message hash value and **z** serves as a private RSA key. Only the distribution centre can calculate **z**, because only the distribution centre knows F(**pq**). The combination of the original hash value h_0 with the private key **z** serves as the basis for validation (verification of integrity) of the new message, as only the distribution centre can create data used to determine a new integrity value **K'** that matches an original integrity value **K**. Next, in step 114, the distribution centre appends the values of **s** and **pq** to the new message. Finally, in step 116 the distribution centre transmits the new message.

Referring now to FIG. 2, a diagram of the new message authentication and validation method according to a preferred embodiment is shown. Each user device (UD) is a specific hardware instance capable of performing computational method steps involved in guaranteeing the integrity of new messages from a distribution centre. Each user device stores in tamper-resistant or completely secure hardware preferably unique values g_1

and g_2 and **a**. In step 202, the user device receives the new message from the distribution centre. Next, in step 204 the user device calculates a test hash value **h** of the new message using the same hashing algorithm employed by the distribution centre. In step 206, the user device proceeds to compute a new integrity value **K'** with the received values of **s** and **pq**, its own device values g_1 and g_2, the hash value of the new message **h**, the digital signet **a** and modulus **M** as described above. In step 208, the new integrity value **K'** is used in further processing as if it were K. Of course, if K' is not equal to K, the further processing will fail. Thus, an attacker trying to gain some advantage by modifying the message will cause total failure rather than the limited behaviour modification desired.

In the preferred embodiment a general purpose computer is programmed according to steps described above. In an alternative embodiment, an article of manufacture - a machine component - is used by a digital processing apparatus to execute the present logic described above.

CLAIMS

1. A method for guaranteeing message integrity, comprising:

transmitting a message together with an encrypted integrity value $\mathbf{h}_0$, said encrypted integrity value $\mathbf{h}_0$ being encrypted with a key based on the hash of said message;

decrypting said encrypted integrity value $\mathbf{h}_0$;

using said integrity value $\mathbf{h}_0$ together with stored integrity values to perform an integrity calculation; and

using the result of said integrity calculation for further processing.

2. The method of claim 1 wherein said integrity calculation is a digital signet calculation.

3. The method of claim 3 wherein results of said digital signet calculation are values used in a content protection scheme.

4. The method of claim 1 wherein said message comprises a software program.

5. A system for guaranteeing message integrity, comprising:

means for transmitting a message together with an encrypted integrity value $\mathbf{h}_0$, said encrypted integrity value $\mathbf{h}_0$ being encrypted with a key based on the hash of said message;

means for decrypting said encrypted integrity value $\mathbf{h}_0$;

means for using said integrity value $\mathbf{h}_0$ together with stored integrity values to perform an integrity calculation; and

means for using the result of said integrity calculation for further processing.

6. A computer program product directly loadable into the internal memory of a digital computer, comprising software code portions for

performing, when said product is run on a computer, a method for guaranteeing message integrity, comprising:

transmitting a message together with an encrypted integrity value $\mathbf{h}_0$, said encrypted integrity value $\mathbf{h}_0$ being encrypted with a key based on the hash of said message;

decrypting said encrypted integrity value $\mathbf{h}_0$;

using said integrity value $\mathbf{h}_0$ together with stored integrity values to perform an integrity calculation; and

using the result of said integrity calculation for further processing.