Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at http://www.giac.org/registration/gsec

Demystifying DSS: The Digital Signature Standard
Richard S. Brehove
GSEC Version 1.4b Option 1

Abstract

This paper examines the requirements of signatures, outlines the technologies involved in creating digital signatures, and describes the components of the Digital Signature Standard (DSS). The basic concepts, not the math, are the focus of the technology investigation. Public key encryption algorithms and secure hash algorithms are explained and discussed. A discussion of the legal aspects of signatures is included. Finally, a comparison is made between paper signatures and digital signatures, showing that digital signatures provided within a well-provisioned Public Key Infrastructure (PKI) satisfy the requirements of signatures at least as well as paper signatures.

Why DSS?

In order to survive and grow, a business will always act in such a way as to improve either its financial position or its reputation. The business is always looking, always investigating for any possibility of getting a hold of the customer before the competition does. Included is not only the product or service that is sold but also the way the company carries out its business transactions. If there is an easier, faster, or cheaper way to do it, a business must find it before the competition.

Many business transactions, either with individuals or other businesses, are executed with a formal document written on paper that each individual to the transaction would sign. After the papers are “drawn-up”, each individual would affix his or her unique “signature” to the paper. This signature would preferably be unique and identify the party as part of the transaction. Anyone who has purchased a home or taken out a substantial loan knows the process and the wrist aches of having to sign documents in an escrow agent’s office for nearly half an hour.

Now that the signature is on the piece of paper, there needs to be a way to identify the individual that signed it, even though the signature may be unreadable. The matching, or “binding”, of the signature to the individual in a business transaction could be done in a number of ways: Signature cards at the bank could be matched with the signature on the document. A Notary service could give assurance that the signatures on the paper were made by a certain individual. Past signatures on other documents could be compared with the one on the document in question. All of these methods have pros and cons, and different methods are used depending on the risk the parties are taking.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

All of this has worked fairly well, for the most part because the business world has learned the risks involved and taken measures that mitigate these to an acceptable level depending on their own perception of risk. In addition, a body of case law has emerged over time, which governs what is acceptable when "signing" a document. In short, people trust the system.

Great. Now enter the computer. The use of the computer in business has evolved from automatic typewriter to word processor to full data repository. Then the next logical step was considered: Instead of the "create, modify, print, sign, and mail or fax" cycle that is usual for a business document, wouldn't it be great to somehow sign the document that is right on the computer screen and forward it to your business partner? Further, wouldn't it be great to have your business partner sign the document right when he gets it in his email, and then immediately return it back to you? Or better yet, have your business customers buy from their PCs immediately by transmitting a signed PO right to your sales office for immediate processing into your system. This could decrease the cost of the transaction and increase the speed of the completion of the transaction. Both mean more money for the business and would tend to improve the financial position.

What we need then is some kind of "digital signature". It needs to have properties that are similar to what we tend to normally think of as a signature. Some of these properties, adapted from the American Bar Association [ABA1] might be as follows:

a) The signature is bound to the signed document. One cannot transfer the signature from one document to another.
b) The signature is bound to the identity of the signer.
c) There is assurance that the signer in fact did see the document that he or she was signing
d) The signer intended to affix his signature to the document being signed
e) The signer agreed to the document and intended to be bound by its contents.
f) The signer cannot deny signing the document at a later time.

Turns out that since 1977 [RSA1] there has been a technology that gets us very close to this for digital documents, as well as any other digital file that might reside on a computer or other digital medium. The enabling technology is Public Key Encryption, or Asymmetric Key Encryption. This technology, together with another technology known as secure hashing, will provide the basic tools to put together a standardized procedure to actually "sign" a digital document. A particular collection of these procedures is called the Digital Signature Standard (DSS). We shall see how close digital signatures get to the above list in this paper.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

An "electronic" signature could take on many forms. It could be a digital representation of a paper signature. It could be simply a notation such as "/s/ Richard Brehove". Or it could even be a "From:" address on an email. These, although they have various levels of validity, do not in general have the properties outlined above and are separate and distinct from the "digital" signatures that are defined by DSS. [ABA1]

Consider the device that the UPS delivery driver gives you to sign for packages being delivered, or the signing pod at the department store where you sign for credit card purchases. These devices and associated software will create a digital representation of your signature and associate this with the transaction in their databases. This is still considered an "electronic signature" even though it seems to have most of the properties outlined above. The primary reason is because there is no reliable way to back-check and verify that the signature is attached to the document. Although these signatures might be considered valid when trying to prove that packages were delivered or a credit card transaction was completed, they do not meet the goals set out for digital signatures, specifically to bind with very high certainty a particular document to a particular key.

Digital signatures need a support infrastructure in order to operate. One particular infrastructure is called a Public Key Infrastructure (PKI). A PKI will define the creation of another cryptographic pile of bits called a Digital Certificate. This is important because the main purpose of a digital certificate is that it will bind the identity of a person (or company or whatever the certificate is for) to a specific key. The certificate is the link between the key and the identity, which we will see. The digital signature, in turn, binds the key to the document being signed.

Legalities

Legally, one might ask how this would compare with a hand-written signature. On the federal level, the Electronic Signatures in Global and National Commerce (E-SIGN) Act became federal law in the United States on October 1, 2000. This simply states that a digital signature cannot be denied legal effect in interstate commerce simply because it is in electronic form. In addition, one type of technology may not be favored over another. This law puts digital signatures on an even par with signatures on paper documents. [DST1]

Governments can specify performance standards when purchasing equipment and software that support digital signatures as long as specific equipment is not stated. They can also specify particular technologies if that is specific to the objective and it is needed to achieve the purpose. [DST1]

Nearly every state has digital signature laws on the books. The first states to blaze this trail were Utah, Washington, Illinois, and Minnesota. These laws were

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

put on the books before E-SIGN was enacted. These laws speak in terms of "asymmetric cryptosystems", "public keys", "private keys", and "digital certificates". [DST1] I anticipate that a body of court law will develop that will clarify and clear up any confusions between state and federal laws if any is required.

Several organizations collaborated to come up with E-SIGN. In the United States, the major organization was the National Conference of Commissioners on Uniform State Laws (NCCUSL). They came up with the Uniform Electronic Transactions Act (UTEA). UTEA is similar to E-SIGN, but is more comprehensive. Twenty-three states have adopted UTEA to govern their e-commerce. UTEA recognizes that a signature can be attributed to a person by considering how the signature was created. Thus, with a good solid infrastructure (the PKI) the digital signature may be awarded equal weight as a handwritten signature on a document. [DST1]

Note that in a court of law, a simple signature on a document is not always good enough. [SCHN1] The signer must testify that he or she signed the document. If the document was notarized, the notarized seal is taken as proof that the signer actually signed the document, or the notary him/herself is called to testify to that effect. The same concepts are still true for digital signatures: The simple verified digital signature on the digital document would not be taken at face value unless there is some evidence that the signature satisfies the requirements of a signature. Having a PKI implemented properly, although it will not give the signature greater legal status, can be that evidence. [DST1]

According to the American Bar Association [ABA1 page 2], A legal signature would have the following properties:

a. A signature should indicate who signed the document and be difficult for another to forge.
b. A signature should indicate what is signed, making it impractical to modify either the signed document or the signature without detection.
c. The act of signing should be an "affirmative act" which establishes the sense and idea that a transaction has just been consummated.
d. The signature and its creation and verification process should provide the greatest assurance of authenticity for the signers and the document without great expenditure of resources.

In the case of businesses, the Uniform Commercial Code (UCC) usually governs the sale of goods. UTEA addresses the UCC and allows most transactions governed by the UCC to be done "electronically". However, it is interesting to note that UTEA specifically excludes the following transactions, which are governed by UCC. According to Davis Law Associates [DAV1], the following are exempt from coverage under UTEA:

- Wills, Trusts, Estates
- Marriage, divorce, adoption and other matters of family law

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

- Commercial paper, notes, loans, drafts and checks
- Bank deposits and collections
- Secured transactions (mortgages, home equity credit loans)
- Court documents and filings
- Any notices terminating or canceling utilities
- Any notices of default, eviction, foreclosure or repossession with respect to a primary residence
- Any notice of termination or cancellation of health or life insurance
- Any product recall notice which may affect health or safety
- Any documentation affecting the chain of possession and handling of hazardous materials, toxic materials or pesticides

These exemptions are understandable. They are low frequency and/or high value transactions. UTEA and other laws like it are mainly on the books to allow e-commerce to thrive and survive. The laws are not a claim of a new trust of computer algorithms. That trust has not been built yet and it will take time. Looks like I'll still have to be in that escrow agents office signing my name over and over for my next home refinance.

The exporting of encryption technology used to be treated as a munition by the Federal Government – just like bullets, tanks, and bombs. Only very weak 40-bit encryption could be exported out of the country. The laws have become more and more relaxed in the past three years. Under the new rules in place since January 2000, you can get an exemption to the export laws (which means your request for an export license will be approved) if you have the NSA do a one time review of the software or product. There is no longer a limit on key length. Exports are still banned to the countries of Cuba, Iran, Iraq, Libya, North Korea, Syria, and Sudan.[EPIC1]

Overview of Public Key (Asymmetric Key) Cryptography

Now let us discuss the technology involved. When we think of encryption, we would normally think of a "secret code" that both the sender and receiver posses that will enable the receiver to decode what the sender encoded. If the sender wanted to send a coded message to another receiver, the sender would presumably need to give the new receiver the secret code (or another secret code and keep track of what code he used for which receiver). This code would have to be kept secret. In this case both sender and receiver share some common information, which is used for both the encryption and decryption of data. The sender will use this information to scramble the message and the receiver will use the same information to unscramble it back into readable form. This is called Secret, or Symmetric Key encryption. Three examples of symmetric key encryption algorithms are Data Encryption Standard (DES) Federal Information Processing Standard (FIPS) 46-3, Advanced Encryption Standard (AES) FIPS 197, and International Data Encryption Algorithm (IDEA).

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

In contrast, the primary characteristic of Public Key Cryptography is the fact that there are two keys instead of just one. These two keys share a specific property in that when one of these keys is used to perform encryption, only the other key is able to decrypt the data.

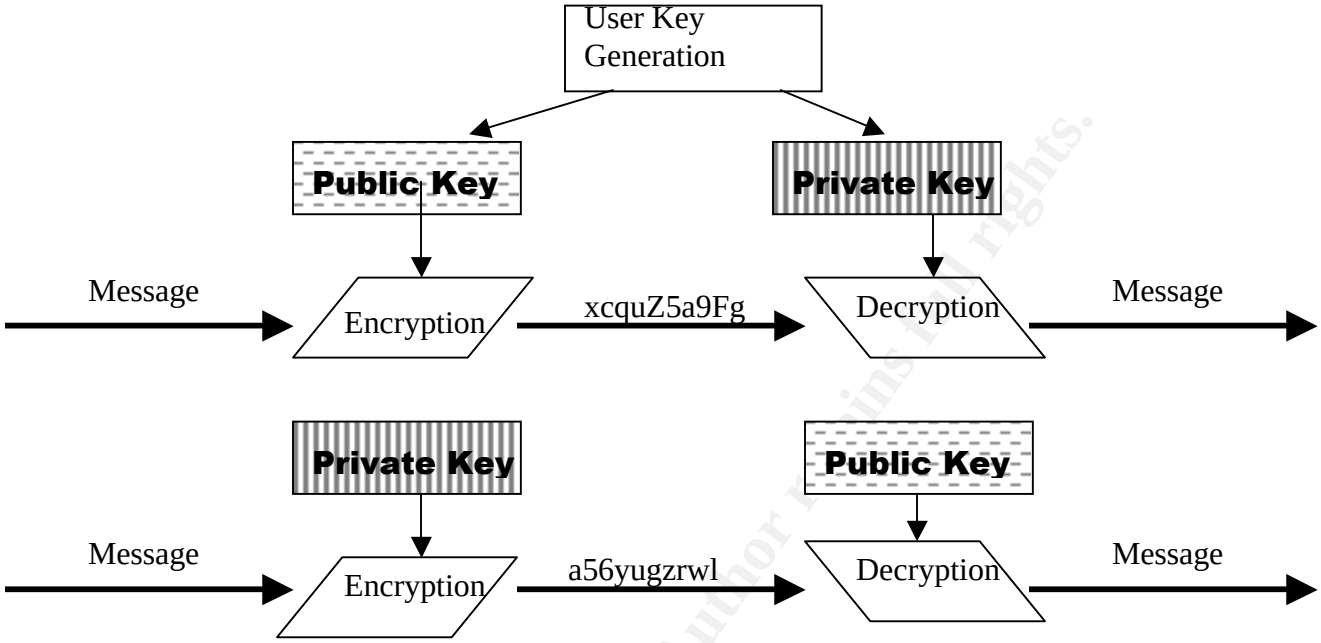

Fig 1. Encryption and Decryption using either the public key or private key

This allows applications that are not possible with symmetric key encryption. Lets say a group of people each generates their two keys. These people would like to exchange encrypted messages. Everyone would take one of their keys and keep it private. We will call this the **private** key. At no time would they ever give this key to anyone or allow anyone access to it. The other key they would freely give to anyone who wants it. We will call this the **public** key. They would perhaps publish it on a server that the whole group had access to. The other property of public key encryption is that the two keys do not seem to have anything to do with each other - the private key cannot be generated or inferred from knowledge of the public key.

If Alice, say, wanted to send a message to all the other people in a group, she could use her own private key to encrypt the message. When the other members received the encrypted message, they would look on the server to get Alice's public key. With this key they could decrypt the message. If this decryption were successful, they would know that the person who had possession of the private key is the only person who could have encrypted that message. Alice would of course know that anyone would be able to decrypt the message – given that they all had access to the server.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

If Alice instead wanted to send a message to only one person in the group, say Bob, she would go to the server and get Bob's public key. Alice would then encrypt the message with Bob's public key. Alice would know that Bob is the only person who can decrypt a message that was encrypted with his public key. When Bob got the message, he would decrypt it using his private key. He of course would have no guarantees that the message came from Alice.

Major Asymmetric Key Cryptosystems

Any algorithm that uses keys as described above are considered Asymmetric Key Algorithms. Some of these algorithms can encrypt whole messages, while others are designed just to be digital signature algorithms. There are four major asymmetric key algorithms: RSA, Diffie-Hellman (DH), Digital Signature Algorithm (DSA), and Elliptic Curve Digital Signature Algorithm (ECDSA). The key sizes for public key algorithms are usually much larger than those of secret key algorithms. Secret key algorithms are usually around 128 to 168 bits, while the public key algorithms use bit sizes from 160 to 1024. The longer key lengths are used for enhanced security, while the shorter key sizes are used for efficiency. [KAUF1 page 134] The mathematics is not immediately understandable to non-cryptanalysts, but a mere user of cryptography should know what mathematical concept the algorithm depends on.

RSA is named after its inventors (Ron Rivest, Adi Shamir, and Leonard Adleman) in 1977 [RSA1] and uses the concept that factoring a big number is hard. It is published as American National Standards Institute (ANSI) X9.31. The key is generated from two large (about 256 bit) prime numbers multiplied together. The product of these two primes is transmitted along with the public key, but the two prime numbers are kept secret and used for the generation of the private key. If anyone had a method of factoring this very large product they would have enough information to generate the private key. [KAUF1 page 135] RSA is used for both encryption of messages and digital signatures. Key sizes for RSA usually range from 512 to 1024 bits or larger.

The Diffie-Hellman (DH) cryptosystem is not really an encryption algorithm, since all it really accomplishes is the exchange of a shared secret key over an unsecured transmission system. But it is still considered a public key system because there is a very large (about 512 bits) prime number and a "magic number" (a number less than the prime number) that is on some public server that is accessible to anyone. To (over)simplify the algorithm, this magic number is then raised to the power of a random number and sent to the other party over the unsecured transmission system. The other party would do the same thing. From this information both can compute the same number. This number they can then use for a secret key to encrypt some data using a secret key encryption algorithm such as DES or IDEA. The security of DH is based on the fact that no one can calculate the secret random number even though they know the initial magic number and the magic number raised to that secret power. This is called

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

computing discrete logarithms. If someone has found out a fast way to do this, they have never told anybody. [KAUF1 page 148]

The DSA (Digital Signature Algorithm) is a public key cryptosystem used only to calculate digital signatures. It was proposed in 1991 by the National Institute of Standards and Technology (NIST) [JOH1 page 4] , blessed by NSA, and specified in the first version of DSS (FIPS 186). It uses keys very similar to DH, but it also includes a computed value (which turns out to be the one half of the DH exchange) as well as the prime number and the "magic number" described for DH. All three of these values constitute the public key. In addition, there is a per-message public key that is sent with every signed message. The DSA is optimized for speed of generating a signature, anticipating its use on low power microprocessors such as those on smart cards or other tokens. Its security, like Diffie-Hellman, also depends on the difficulty of finding discrete logarithms. [KAUF1 page 152-155] If ever we read in the news that someone has found a fast way to compute discrete logarithms, the paper you are reading now would be a history paper.

ECDSA (Elliptic Curve Digital Signature algorithm) is a relative newcomer on the encryption scene and is also used only to calculate signatures. It became an ISO standard in 1998, an ANSI standard X9.62 in 1999, and it became a NIST standard (FIPS 186-2, the DSS) in 2000. It has however been around and considered for cryptographic use since 1985. [JOH1 page 2] Being a newcomer means that there hasn't been as many hours of smart people trying to break the algorithm. The advantages it brings are smaller bit size (for the same security value) and faster implementation. The math here is even more intense, but elliptic curve algorithms involve adding points on an ellipse. The secret key is trying to find out what you added to one point to get the other. No, you don't just subtract. This is analogous to the DSA problem of finding out what the power was you raised a number to in order to get this other number. It is probably more secure. Why? Although finding discrete logarithms are hard, there are some algorithms, called "Pollard's rho" algorithms [JOH1 page 5][JOH2 page 11] and "number field sieve" algorithms [GOR1] that can begin to find discrete logarithms if only there were lots of time and computing power available. This time is dependent on key size. There are similar algorithms for elliptic curves, [JOH1 page 29] but they take orders of magnitude longer. So the claim is made that the security of DSA at 1024 bits is equivalent to ECDSA at 160 bits. [JOH2 page 8] Key sizes for ECDSA algorithms in implementation usually range from 160 to 180 bits.

Public Key Algorithms, then, give us the ability to share secrets selectively and independently. They work because the two keys are related in a mathematical way as inverses of each other. One number will do in one algorithm what the "inverse" of the number will undo in another algorithm. Their security is guaranteed by intractable math problems.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

It is also important to note here that public key algorithms are typically much slower than secret key algorithms. When a message is encrypted with the RSA algorithm, it is usually a secret key that is being encrypted with the algorithm. The actual message itself is probably being encrypted with a much faster secret key algorithm, such as DES or 3DES (Triple-DES – the DES algorithm run three times in succession). To decrypt the message, the receiver would first use RSA to decrypt the secret key and then use that key to decrypt the message using DES or 3DES. Further, when we speak of "signing the message", in actual practice the data that is being signed is a cryptographic hash of the message which is much smaller than the actual message. This takes us to our next subject.

Secure Hashes

A "checksum" is a simple way of taking a very long string of data, which could be (almost) any length, and reducing it down to a specific number of bits, such that changing any bit of the original data would change the checksum. Hashing is the same concept, and it has been part of computing for a very long time. Hashing takes a message of any length and condenses it, or digests it, into a specified number of bits. It is a one-way function – you cannot go backwards and find the message from the hash. Why would you want to do this to a message? If the sender of a message calculated a hash before transmission, and then sent the hash along with the message, the receiver of the message could verify that the message was not garbled in transmission if his own calculation of the hash matched that of the senders' calculation.

A secure hash, or Message Digest, has additional properties. As you might guess, a message digest uses cryptographic algorithms to provide these additional properties: [KAUF1, Chapter 4]

1. Changing any bit of the original message will unpredictably change each and every bit of the message digest. There is no information you can get from the message digest about the original message.
2. It is not practical to find two messages with the same message digest (called a **collision**).
3. It is not practical to find a message with a given message digest.

The two most popular Message Digest algorithms are MD5 (Message Digest 5) and SHA-1 (Secure Hash Algorithm –1). Both algorithms are very similar to each other and take the same mathematical approach. Ron Rivest developed MD5 in 1991. There were previous versions MD2 and MD4 released, but these have been outdated – MD2 because it was too slow and MD4 because, although it was designed for speed, had a few problems that made the experts nervous. MD5 addressed these problems at the cost of speed and is considered more secure. [KAUF1, page 106] MD4 should be considered "broken" [CRYP1] (The

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

word “broken” in crypto-speak means that too many smart people have found real or theoretical problems with the crypto algorithm, and it just so happens that there is another algorithm available to fix the problems). MD5 will take an arbitrary message (less than a length of 2^64 bits) and create a 128-bit digest. [KAUF1, page 123] [RIV1]

SHA-1 was developed by NIST as an update to SHA in 1994. The latest version is dated August 1, 2002 and is published as the Secure Hash Standard (SHS) FIPS 180-2. It takes an arbitrary message and creates a 160-bit digest. SHA-1 is slower than MD5, but since the hash is 160 bits instead of MD5’s 128 bits it is considered more secure to brute force attacks. [HASH1] The Secure Hash Standard actually specifies four hashing algorithms: SHA-1, SHA-256, SHA-384, and SHA-512. The main differences are in the size of the largest message it can hash (up to 2^128 bits) and the message digest size (from 160 to 512 bits). [180-2].

Secure Hash algorithms work because no one has been able to deny them the three properties outlined above. Note that this is a different problem than public key encryption. With hashes we do not care about finding the message again after we “digest” it. There are no keys to keep secret. The message just needs to be scrambled enough such that the hash is unrecognizable. The properties above are important if we are to guarantee that no one has manipulated the messages. This is important to digital signatures and something that simple checksums do not give.

The message digest is always the same number of bits. One might wonder how a message digest can represent the message, since surely more messages can be created than can be represented in a finite number of bits. The answer is first of all that there are 2^160 possible SHA-1 “fingerprints”. The chances of a collision are low. Second, in order find two messages with the same 160-bit message digest, one would have to go through approximately 2^80 messages. This is considered infeasible given the current state of the art. The secure hash can be considered the “fingerprint” of the message. [KAUF1, page 102]

Digital Certificates

We have already discussed the need for a Public Key Infrastructure (PKI) to support digital signatures. In our previous example, if Bob wanted to get Alice’s public key for a signed document he just received, Bob would need to have some assurance that this particular public key does in fact belong to Alice. Without this assurance, the only thing Bob can link is the document to a private key simply by the fact that the decryption with the public key worked. There is no assurance of who owns the private key.

If Bob were to get the public key from a reliable trusted source, he could then make the link from the key to Alice herself. Alice could walk over to Bob’s office

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

and hand him her public key on a floppy disk, but a more general solution is required. Let's say the bank that Alice and Bob both bank at, for example, kept all the public keys, and both Alice and Bob and anyone else who wanted in on this exclusive club walked over to the bank and handed them their keys on floppy disks (or transferred them encrypted electronically by, perhaps, a DH exchange and 3DES encryption). The bank would know all about Alice and can vouch for her identity, address, work, etc. The bank could then take Alice's public key and some information about Alice, like her email address, and encrypt this information with the bank's own private key. This information, then, becomes Alice's Digital Certificate, which has been "signed" by the bank and can be freely published on servers. Now, when Bob wants to decrypt Alice's signature on a message he just received, he would go get Alice's Digital Certificate from the bank's server. Since Bob received the bank's public key from a reliable source, probably when he signed up for the service, he can decrypt the certificate and retrieve Alice's public key, along with other information about Alice. Bob can now have assurance that the person who owns the private key for this particular public key is indeed Alice, since a third party (the bank) went to the trouble of verifying Alice's identity, and Alice's public key has been "bound" to her identity with the digital certificate obtained from the bank. If the retrieved public key works, Bob has a very highly reliable way of determining that the message actually came from Alice – or Alice actually signed the document. It is the certificate, therefore, that "binds" the identity of Alice to Alice's keys.

A Digital Certificate is only one part of what an entire PKI will generate. A PKI also gives a way to solve other problems that come up, like keys getting lost or stolen, time limited keys, keys being deemed revoked due to people changing responsibilities, and generation and distribution of new keys. The DSS does not specify any PKI functions nor does it deal with the authentication of public keys. We will need these guarantees however if digital signatures are to meet the properties for digital signatures that we have already discussed.

Digital Signatures and the DSS Algorithm

The tools are now in place to describe digital signatures. Simply, to sign a document one just needs to run the document through a message digest algorithm creating a "fingerprint" of the document, and then use this bit sequence in the signing procedure of the public key algorithm selected. Signing a document encrypts using the private keys. The resulting bit sequence is transmitted right along with the original document. The receiver, wishing to verify the signature, will get the sender's public key from a server (or from a certificate as described above), calculate the message digest of the message the same way the sender did, and use this information in the verification procedure of the decryption algorithm. If the algorithm indicates successful verification, the signature is considered valid. Validity would mean:

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

1. The message was not altered or modified between the time it was signed and the time the signature was verified.
2. The private key associated with this public key is the only key that could have signed this document.

If the public keys were retrieved from a digital certificate (a PKI is in place), the keys can be traced to an identity, given that the source of the certificate (the bank for instance) is trustworthy. One more item could then be known.

3. The owner of this private key is the person (or company) who signed this document. Unless the secret key was compromised, there would be no denying this.

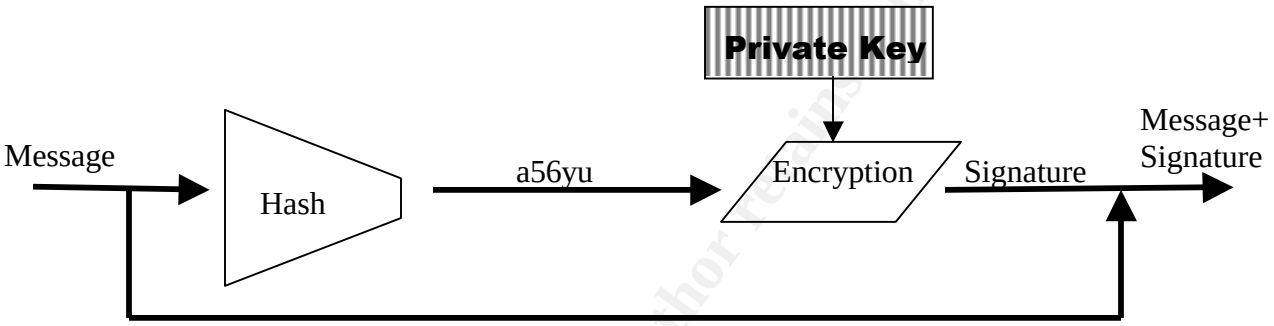

Fig 2. Creation of a Digital Signature

DSS is a cryptographic system for generating and verifying digital signatures. The latest version of DSS is FIPS 186-2 dated January 27, 2000. It was originally published in 1994 with only DSA as the approved public key algorithm. Since FIPS 186 was first published, two revisions of the standard have been approved to also permit the use of RSA, as specified in American National Standard Institute (ANSI) X9.31 and the Elliptic Curve Digital Signature Algorithm (ECDSA), as specified in ANSI X9.62, for generating digital signatures. For the secure hash, the DSS specifies FIPS 180-2, Secure Hash Standard. Since this is a FIPS standard, it is applicable to all federal agencies for sensitive unclassified information. They should use it unless they have a good reason not to use it – in which case they can get a waiver. The DSS gives details of how to use each one of the public key algorithms, giving ranges for the required parameters, and even gives recommended elliptic curves for ECDSA. [186-2]

The DSS does not specify the whole PKI infrastructure - generation of the digital certificates, a method of revoking certificates and keys and how the public keys and certificates are distributed and managed. Clearly this is needed in order to provide validity to the keys being used as specified in the DSS. Specifying a PKI will allow digital signatures to achieve the requirements set out when we began this investigation.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

DSS also requires a procedure for generation of prime numbers and generation of pseudo-random numbers, both of which are specified in DSS. A DSS signature will be 320 bits in length when DSA is used with the 160-bit SHA-1.

Of the major security functions of Privacy, Integrity, Authentication, Authorization, Access, Availability, and Non-Repudiation, we are looking for digital signatures to give us Authentication, Integrity, and Non-Repudiation. The secure hash provides integrity, which is protected by the encryption function. The association within a PKI digital certificate linking a verified identity to a key provides authentication. Non-Repudiation is provided by the inability for anyone other than the holder of the private key to create a particular sequence of bits given a particular message. Taken all together, we have a particular digital document directly traceable to a particular individual.

Applications

The DSS specification itself identifies many applications that can benefit from cryptographically secure digital signatures. Any application that can benefit from the integrity of the message and authenticity of the sender can use DSS. It can be used in software distribution, data storage, and database integrity. Applications such as electronic mail, electronic funds transfer, and electronic data interchange can also benefit from the non-repudiation features of DSS. Digital signatures will also be a great enabler of e-commerce applications, giving merchants more ways to sell products and services. [186-2]

Is it a good signature?

Finally, let us return to our original list of signature characteristics and examine how digital signatures compare with paper signatures. In this discussion, we will assume that the digital signatures are functioning within a well-provisioned PKI.

The signature is bound to the signed document for both digital signatures and paper signatures. The digital signature is bound with a very high degree of confidence given the current state of the art in mathematics, and is very easy for anyone to verify. Given the trust in the notary or other witnesses, a paper signature is also very tightly bound to the signed document. Further, if the signature moved from one document to the other, this would cause the DSS to fail validation (The SHA-1 hash would be different) and in the case of the paper signature there would need to be specific techniques employed to detect the fraud.

The signature is bound to the identity of the signer via either the digital certificate or the notary testimony. Both are very strong methods that link a particular signature to a particular document – one by human testimony and the other by mathematics. The trust in the digital certificate is a trust in both the PKI

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

infrastructure and in the signing computer itself. For paper signatures, the identity can be established simply by trusting the notary.

Did the signer actually see the document, intend to affix his/her signature to that document, and agree to be bound by its contents? With the paper signature there is an expectation that all this is true. There is a direct link between the signer and the document – all that comes between the two is a pen. For digital signatures, however, a whole computer comes between the signer and the document. Unless the computer is trusted, the entity that actually digitally signed the document would be the computer rather than the individual at the keyboard. With an untrusted computer, there is no telling what document the computer actually signed and transmitted. The secret key is after all stored on the computer, and rogue versions of programs can falsely sign documents that are never viewed by the user. [SCHN1] Worse, they can send the private keys anywhere on the network. [KAN1] This does not render digital signatures useless by any means. This is just another risk that the industry will need to mitigate before digital signatures are fully trusted. With most transactions verifying the transaction to the computer rather than the user is good enough. It is up to the signer to take steps to assure him/herself that the computer that is used for signing documents is virus and trojan horse free, and that the password used to sign the documents is kept secret. In the paper world, it is also the signer's responsibility to, for example, keep unused checks in a safe place. If any of this gets compromised, there should be steps in each infrastructure to deal with the problem.

The signer cannot deny signing the document depending on how tightly the signature is bound to the identity of the signer. For paper signatures, a binding perhaps is the notary. For digital signatures, the binding is the digital certificate.

Conclusions

Digital signatures compare favorably with paper signatures in satisfying requirements for signing a document. Both technologies must deal with trust issues and create ways in which to mitigate each and every risk that is perceived. Neither method is foolproof. The legal community is dealing with the problems of digital signatures and is beginning to set overall parameters. This will give businesses a structure to move forward on projects requiring digital signatures with at least some confidence. Perhaps, with growth in the trust of digital signatures – guarantees that the signatures will be accepted and methods for resolving disputes in their use – we will see growth in the use of them as well.

References

[180-2] Federal Information Processing Standard (FIPS) Publication 180-2 *Secure Hash Standard (SHS).* US/DoC NIST. August 1, 2002.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

[186-2] Federal Information Processing Standard (FIPS) Publication 186-2 *Digital Signature Standard (DSS).* US/DoC NIST. January 27, 2000 with Change Notice #1 October 5, 2001.

[ABA1] American Bar Association, Section of Science and Technology, Information Security Committee. "Digital Signature Guidelines Tutorial". URL: http://www.s2.chalmers.se/iths/pdf/Digital%20signature%20tutorial.pdf (22 June 2003).

[CRYP1] The Cryptix Foundation Limited and David Hopwood, copyright 1995-2002 . "Standard Cryptographic Algorithm Naming". URL: http://www.users.zetnet.co.uk/hopwood/crypto/scan/md.html (23 June 2003).

[DAV1] Davis Law Associates CEO Roundtable Notes June 2000. URL: http://www.sandiegobusinesslaw.com/pdf/ceoroundtable-jun2000.pdf (22 June 2003).

[DST1] Digital Signature Trust Co. "Industry Legal News". URL: http://www.digsigtrust.com/support/industry_legal_news.html (22 June 2003).

[EPIC1] Electronic Privacy and Information Center (EPIC). "Cryptography and Liberty 2000 An International Survey of Encryption Policy". URL: http://www2.epic.org/reports/crypto2000/countries.html#Heading133 (22 June 2003).

[GOR1] Gordon, Daniel M. "Discrete Logarithms in GF(p) Using the Number Field Sieve". Department of Computer Science, University of Georgia, Athens, GA. Feburary 24, 1992.

[HASH1] Hash Algorithm Directory. "The Secure Hash Algorithm Directory MD5, SHA-1, HMAC". URL: http://www.secure-hash-algorithm-md5-sha-1.co.uk/index.htm (22 June 2003).

[JOH1] Johnson, Don and Menezes, Alfred and Vanzone, Scott. "The Elliptic Curve Digital Signature Algorithm". Certicom Research, Canada and Department of Combinatorics and Optimization, University of Waterloo, Canada Certicom Corporation 2001.

[JOH2] Johnson, Don and Menezes, Alfred. "Elliptic Curve DSA (ECDSA) An Enhanced DSA". Certicom Corp. and Auburn University URL: http://www.qrst.de/html/dsds/ec/ECDSA.pdf (23 June 2003).

[KAN1] Kaner, Cern. "The Insecurity of the Digital Signature". September 1997 URL: http://www.badsoftware.com/digsig.htm (22 June 2003).

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

[KAUF1] Kaufman, Charlie; Perlman Radia; Mike Speciner. Network Security, Private Communication in a Public World. Upper Saddle River, New Jersey, Prentice Hall, 1995.

[RSA1] RSA, Inc. "RSA-based Cryptographic Schemes". Copyright 2003. URL: http://www.rsasecurity.com/rsalabs/rsa_algorithm/index.html (22 June 2003).

[RIV1] Rivest, Ron. "The MD5 Message Digest Algorithm", RFC 1321. MIT and RSA Data Security, Inc. April 1992.

[SCHN1] Schneier, Bruce. "Why Digital Signatures are not Signatures". copyright 2002. Counterpane Internet Security Inc. URL: http://www.hipaadvisory.com/tech/whynot.htm (22 June 2003).

© SANS Institute 2003, Author retains full rights.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.