Lecture 10: Public key cryptography

Boaz Barak

People have been dreaming about heavier than air flight since at least the days of Leonardo Da Vinci (not to mention Icarus from the greek mythology). Jules Vern wrote with rather insightful details about going to the moon in 1865. But, as far as I know, in all the thousands of years people have been using secret writing, until about 50 years ago no one has considered the possibility of communicating securely without first exchanging a shared secret key. However, in the late 1960's and early 1970's, several people started to question this "common wisdom".

Perhaps the most surprising of these visionaries was an undergraduate student at Berkeley named Ralph Merkle. In the fall of 1974 he wrote a project proposal for his computer security course that while "it might seem intuitively obvious that if two people have never had the opportunity to prearrange an encryption method, then they will be unable to communicate securely over an insecure channel… I believe it is false". The project proposal was rejected by his professor as "not good enough". Merkle later submitted a paper to the communication of the ACM where he apologized for the lack of references since he was unable to find any mention of the problem in the scientific literature, and the only source where he saw the problem even *raised* was in a science fiction story. The paper was rejected with the comment that "Experience shows that it is extremely dangerous to transmit key information in the clear." Merkle showed that one can design a protocol where Alice and Bob can use T invocations of a hash function to exchange a key, but an adversary (in the random oracle model, though he of course didn't use this name) would need roughly T^2 invocations to break it. He conjectured that it may be possible to obtain such protocols where breaking is *exponentially harder* than using them, but could not think of any concrete way to doing so.

We only found out much later that in the late 1960's, a few years before Merkle, James Ellis of the British Intelligence agency GCHQ was having similar thoughts. His curiosity was spurred by an old World-War II manuscript from Bell labs that suggested the following way that two people could communicate securely over a phone line. Alice would inject noise to the line, Bob would relay his messages, and then Alice would subtract the noise to get the signal. The idea is that an adversary over the line sees only the sum of Alice's and Bob's signals, and doesn't know what came from what. This got James Ellis thinking whether it would be possible to achieve something like that digitally. As he later recollected, in 1970 he realized that in principle this should be possible, since he could think of an hypothetical black box B that on input a "handle" α and plaintext p would give a "ciphertext" c and that there would be a secret key β corresponding to α, such that feeding β and c to the box would recover p. However, Ellis had no idea how to actually instantiate this box. He and others kept giving this question as a puzzle to bright new recruits until one of them, Clifford Cocks, came up in 1973 with a candidate solution loosely based on the factoring problem; in 1974 another GCHQ recruit, Malcolm Williamson, came up with a solution using modular exponentiation.

But among all those thinking of public key cryptography, probably the people who saw the furthest were two researchers at Stanford, Whit Diffie and Martin Hellman. They realized that with the advent of electronic communication, cryptography would find new applications beyond the military domain of spies and submarines. And they understood that in this new world of many users and point to point communication, cryptography will need to scale up. They envisioned an object which we now call "trapdoor permutation" though they called "one way trapdoor function" or sometimes simply "public key encryption". This is a collection of permutations $\{p_k\}$ where p_k is a permutation over (say) $\{0,1\}^{|k|}$, and the map $(x,k) \mapsto p_k(x)$ is efficiently computable *but* the reverse map $(k,y) \mapsto p_k^{-1}(y)$ computationally hard. Yet, there is also some secret key $s(k)$ (i.e., the "trapdoor") such that using $s(k)$ it is possible to efficiently compute p_k^{-1}. Their idea was that using such a trapdoor permutation, Alice who knows $s(k)$ would be able to publish k on some public file such that every one who wants to send her a message x could do so by computing $p_k(x)$. (While today we know, due to the work of Goldwasser and Micali, that such a deterministic encryption is not a good idea, at the time Diffie and Hellman had amazing intuitions but didn't really have proper definitions of security.) But they didn't stop there. They realized that protecting the *integrity* of communication is no less important than protecting its *secrecy*. Thus they imagined that Alice could "run encryption in reverse" in order to certify or *sign* messages. That is, given some message m, Alice would send the value $x = p_k^{-1}(h(m))$ (for a hash function h) as a way to certify that she endorses m, and every person who knows k could verify this by checking that $p_k(x) = h(m)$.

However, Diffie and Hellman were in a position not unlike physicists who predicted that a certain particle should exist but without any experimental verification. Luckily they met Ralph Merkle, and his ideas about a probabilistic *key exchange protocol*, together with a suggestion from their Stanford colleague John Gill, inspired them to come up with what today is known as the *Diffie Hellman Key Exchange* (which unbeknownst to them was found two years earlier at GCHQ by Malcolm Williamson). They published their paper "New Directions in Cryptography" in 1976, and it is considered to have brought about the birth of modern cryptography. However, they still didn't find their elusive trapdoor function. This was done the next year by Rivest, Shamir and Adleman who came up with the RSA trapdoor function, which through the framework of Diffie and Hellman yielded not just encryption but also signatures (this was essentially the same function discovered earlier by Clifford Cocks at GCHQ, though as far as I can tell Cocks, Ellis and Williamson did not realize the application to digital signatures). From this point on began a flurry of advances in cryptography which hasn't really died down till this day.

Public Key Cryptography: Definition

We now discuss how we define security for public key cryptography. As mentioned above, it took quite a while for cryptographers to arrive at the "right" definition, but in the interest of time we will skip ahead to what by now is the standard basic notion:

Definition A triple of efficient algorithms (G, E, D) is a *public key encryption scheme* if it satisfies the following:

- G is a probabilistic algorithm known as the *key generation algorithm* that on input 1^n outputs a distribution over pair of keys (e, d).
- For every $m \in \{0,1\}^n$, with probability $1 - negl(n)$ over the choice of $(e, d) = G(1^n)$ and the coins of E,D, $D_d(E_e(m)) = m$.

We say that (G, E, D) is *CPA secure* every efficient adversary A wins the following game with probability at most $1/2 + negl(n)$:

- $(e, d) = G(1^n)$
- A is given e and outputs a pair of messages $m_0, m_1 \in \{0,1\}^n$.
- A is given $c = E_e(m_b)$ for $b \leftarrow_R \{0,1\}$.
- A outputs $b' \in \{0,1\}$ and *wins* if $b' = b$.

Note that we don't explicitly give A access to the encryption oracle since it can compute it on its own using e.

One metaphor for a public key encryption is a "self-locking lock" where you don't need the key to *lock it* (but rather you simply push the shackle until it clicks and lock) but you do need the key to *unlock* it. So, if Alice generates $(e, d) = G(1^n)$ then e serves as the "lock" that can be used to *encrypt* messages for Alice while only d can be used to *decrypt* the messages. Another way to think about it is that e is a "hobbled key" that can be used for only some of the functions of d.

The obfuscation paradigm

Why would someone imagine that such a magical object could exist? The writing of both James Ellis as well as Diffie and Hellman suggests that their thought process was roughly as follows. You imagine a "magic black box" B such that if all parties have access to B then we could get a public key encryption scheme. Now if public key encryption was impossible it would mean that for every possible program P that computes the functionality of B, if we distribute the code of P

to all parties then we don't get a secure encryption scheme. That means that *no matter what program P the adversary gets*, she will always be able to get some information out of that code that helps break the encryption, even though she wouldn't have been able to break it if P was a black box. Now intuitively understanding arbitrary code is a very hard problem, so Diffie and Hellman imagined that it might be possible to take this ideal B and compile it to some sufficiently low level assembly language so that it would behave as a "virtual black box". In particular, if you took, say, the encoding procedure $m \mapsto p_k(m)$ of a block cipher with a particular key k, and ran it through an optimizing compiler you might hope that while it would be possible to perform this map using the resulting executable, it will be hard to extract k from it, and hence could treat this code as a "public key". Diffie and Hellman couldn't really find a way to make this work, but it convinced them this notion of public key is not *inherently impossible.* This concept of compiling a program into a functionally equivalent but "inscrutable" form is known as *software obfuscation* . It had turned out to be quite a tricky object to both define formally and achieve, but it serves as a very good intuition as to what can be achieved, even if, as the random oracle, this intuition can sometimes be too optimistic. (Indeed, if software obfuscation was possible then we could obtain a "random oracle like" hash function by taking the code of a function f_k chosen from a PRF family and compiling it through an obfuscating compiler.)

We will not formally define obfuscators yet, but on intuitive level it would be a compiler that takes a program P and maps into a program P' such that:

- P' is not much slower/bigger than P (e.g., as a Boolean circuit it would be at most polynomially larger)
- P' is functionally equivalent to P, i.e., $P'(x) = P(x)$ for every input x.[1]
- P' is "inscrutable" in the sense that seeing the code of P' is not more informative than getting *black box access* to P.

Let me stress again that there is no known construction of obfuscators achieving something similar to this definition. In fact, the most natural formalization of this definition is *impossible* to achieve (as we might see later in this course). However, when trying to stretch your imagination to consider the amazing possibilities that could be achieved in cryptography, it is not a bad heuristic to first ask yourself what could be possible if only everyone involved had access to a magic black box. It certainly worked well for Diffie and Hellman.

Some concrete candidates:

We would have loved to prove a theorem of the form:

"Theorem": If the PRG assumption is true then there exists a CPA-secure public key encryption.

[1]For simplicity, assume that the program P is *side effect free* and hence it simply computes some function, say from $\{0,1\}^n$ to $\{0,1\}^\ell$ for some n, ℓ.

This would have meant that we do not need to assume anything more than the already minimal notion of pseudorandom generators (or equivalently, one way functions) to obtain public key cryptography. Unfortunately, no such result is known. The kind of results we know have the following form:

Theorem: If problem X is hard then there exists a CPA-secure public key encryption.

Where X is some problem that people have tried to solve and couldn't. Thus we have various *candidates* for public key encryption and we fervently hope that at least one of them is actually secure. The dirty little secret of cryptography is that we actually don't have that many candidates. We really have only two well studied families.[2] One is the "group theoretic" family that relies on the difficulty of the discrete logarithm (over modular arithmetic or elliptic curves) or the integer factoring problem. The other is the "coding/lattice theoretic" family that relies on the difficulty of solving noisy linear equations or related problems such as finding short vectors in a *lattice* and solving instances of the "knapsack" problem. Moreover, problems from the first family are known to be *efficiently solvable* in a computational model known as "quantum computing". If large scale physical devices that simulate this model, known as *quantum computers*, then they could break all cryptosystems relying on these problems.

We will start by describing cryptosystems based on the first family (which was discovered before the other, as well as much more widely implemented), and in future lectures talk about the second family.

Diffie Hellman Encryption (aka El-Gamal)

The Diffie-Hellman public key system is built on the presumed difficulty of the *discrete logarithm problem*:

For any number p, let $\mathbb{Z}_p$ be the set of numbers $\{0, \ldots, p-1\}$ where addition and multiplication are done modulo p. We will think of numbers p that are of magnitude roughly 2^n, so they can be described with about n bits. We can clearly multiply and add such numbers modulo p in $poly(n)$ time. If $g \in \mathbb{Z}_p$ and a is any natural number, we can define g^a to be simply $g \cdot g \cdots g$ (a times). A priori one might think that it would take $a \cdot poly(n)$ time to compute g^a, which might be exponential if a itself is roughly 2^n. However, we can compute this in $poly(\log a) \cdot n)$ time using the *repeated squaring trick*. The idea is that if $a = 2^\ell$ then we can compute g^a in ℓ by squaring g ℓ times, and a general a can be decomposed into powers of two using the binary representation. The *discrete logarithm* problem is the problem of computing, given $g, h \in \mathbb{Z}_p$, a number a such that $g^a = h$. If such a solution a exists then there is always also a solution of size at most p (can you see why?) and so the solution can

[2] There have been some other more exotic suggestions for public key encryption (including some by yours truly as well as suggestions such as the isogeny star problem , though see also this), but they have not yet received wide scrutiny.

be represented using n bits. However, currently the best known algorithm for computing the discrete logarithm run in time roughly $2^{n^{1/3}}$ which currently becomes prohibitively expensive when p is about 2048 bits.[3]

John Gill suggested to Diffie and Hellman that modular exponentiation can be a good source for the kind of "easy-to-compute but hard-to-invert" functions they were looking for. Diffie and Hellman based a public key encryption scheme as follows:

- The *key generation algorithm*, on input n, samples a prime number p of n bits description (i.e., between 2^{n-1} to 2^n), a number $g \leftarrow_R \mathbb{Z}_p$ and $a \leftarrow_R \{0, \ldots, p-1\}$. We also sample a hash function $H : \{0,1\}^n \rightarrow \{0,1\}^\ell$. The public key e is (p, g, g^a, H) while the secret key d is a.[4]
- The *encryption algorithm*, on input a message $m \in \{0,1\}^\ell$ and a public key $e = (p, g, h, H)$ will choose a random $b \leftarrow_R \{0, \ldots, p-1\}$, and output $(g^b, H(h^b) \oplus m)$.
- The *decryption algorithm*, on input a ciphertext (f, y) and the secret key, will output $H(f^a) \oplus y$.

The correctness of the decryption algorithm follows from the fact that $(g^a)^b = (g^b)^a = g^{ab}$ and hence $H(h^b)$ computed by the encryption algorithm is the same as the value $H(f^a)$ computed by the decryption algorithm. A simple relation between the discrete logarithm and the Diffie-Hellman system is the following:

Lemma: If there is a polynomial time algorithm for the discrete logarithm problem then the Diffie-Hellman system is *insecure*.

Proof: Using a discrete logarithm algorithm, we can compute the private key a from the parameters p, g, g^a present in the public key, and clearly once we know the private key we can decrypt any message of our choice. QED

Unfortunately, no such result is known in the other direction. However in the random oracle model, we can prove that this protocol is secure assuming the task of computing g^{ab} from g^a and g^b (which is now known as the *Diffie Hellman problem*) is hard.[5]

[3]If the time was truly $2^{n^{1/3}}$ then you might think we'd need numbers of the order of $128^3 \approx 2 \cdot 10^6$ bits to get 128 bits of security, while NIST estimates that we only need a 3072 bit key to get this level of security. This is because of the constants (and some slow growing non constant functions) in the exponent of the running time of the current best algorithm. However, you can see that the algorithm runs better than $2^{constant \cdot n}$ in the fast that to double the estimated security to 256 bits, NIST recommends that we multiply the RSA keysize 5-fold to $15,360$. (The same document says that SHA-256 gives 256 bits of security as a pseudorandom generator but only 128 bits when used to hash documents for digital signatures; do you know why?)

[4]Formally the secret key should also contain all information in the public key, but we omit this for simplicity of notation.

[5]One can get security results for this protocol without a random oracle if we assume a stronger variant known as the *Decisional Diffie Hellman (DDH)* assumption.

Computational Diffie Hellman Assumption: Let $\mathbb{G}$ be a group elements of which can be described in n bits, with an associative and commutative multiplication operation the can be computed in $poly(n)$ time. The _Computational Diffie Hellman (CDH)___ assumption holds for $\mathbb{G}$ if for every generator (see below) g of $\mathbb{G}$ and efficient algorithm A, the probability that on input g, g^a, g^b, A outputs the element g^{ab} is negligible.

Theorem: The Diffie Hellman system for $\mathbb{G}$ is CPA secure in the random oracle model whenever CDH holds for $\mathbb{G}$.

Proof: For CPA security we need to prove that the following two distributions are computationally indistinguishable for every $m \neq m'$ (can you see why?)

- $(g^a, g^b, H(g^{ab}) \oplus m)$
- $(g^a, g^b, H(g^{ab}) \oplus m')$

By the hybrid argument, this follows from showing that the following two distributions are computationally indistinguishable:

- $(H, g, g^a, g^b, H(g^{ab}))$
- (H, g, g^a, g^b, U_ℓ)

But an adversary will only be able to distinguish between these two cases if he makes the query g^{ab} to the random oracle $H(\cdot)$ (as otherwise the value of this query is completely uniform), and so if a T query adversary A distinguishes between these two distributions with probability ϵ then it can be converted into a CDH algorithm that succeeds with probability roughly ϵ/T. QED

> **Elliptic curve cryptography:** As mentioned, the Diffie Hellman systems can be run with many variants of Abelian groups. Of course, for some of those groups the discrete logarithm problem might be easy, and so they would be inappropriate to use for this system. One variant that has been proposed is elliptic curve cryptography. This is a group consisting of points of the form $(x, y, z) \in \mathbb{Z}_p^3$ that satisfy a certain equation, and multiplication can be defined according in a certain way. The main advantage of elliptic curve cryptography is that the best known algorithms run in time $2^{\approx n}$ as opposed to $2^{\approx n^{1/3}}$ which allows for much shorter keys. Unfortunately, elliptic curve cryptography is just as susceptible to quantum algorithms as the discrete logarithm problem over $\mathbb{Z}_p$.

> **Encryption vs Key Echange and El Gamal:** In most of the cryptography literature the protocol above is called the *Diffie Hellman Key Exchange* protocol, and when considered as a public key system it is sometimes known as *ElGamal encryption*.[6] The reason for this mostly stems from the early confusion on what are the right

[6]ElGamal's actual contribution was to design a *signature scheme* based on the Diffie-Hellman problem, a variant of which is the Digital Signature Algorithm (DSA) described below.

security definitions. Diffie and Hellman thought of encryption as a *deterministic* process and so they called their scheme a “key exchange protocol”. They also envisioned that public key encryption could be just as efficient as private key encryption. The work of Goldwasser and Micali showed that encryption must be probabilistic for security, and efficiency considerations also imply that public key encryption is always used today just as a mechanism to exchange a key to be used as a private key. Together this means that there is not much point to distinguish between a two message key exchange algorithm and a public key encryption.

A little bit of group theory.

If you haven’t seen group theory, it might be useful for you to do a quick review. We will not use much group theory and mostly use the theory of finite commutative (also known as Abelian) cyclic groups which are such a baby version that it might not be considered true “group theory” by many group theorists. What you need to remember is the following:

- $\mathbb{G}$ is a finite commutative group is a finite set together with a multiplication operation that satisfies $a \cdot b = b \cdot a$ and $(a \cdot b) \cdot c = (a \cdot b) \cdot c)$.
- $\mathbb{G}$ has a special element known as 1, where for every $g \in \mathbb{G}$ there exists an element $g^{-1} \in \mathbb{G}$ such that $gg^{-1} = 1$.
- For every $g \in \mathbb{G}$, the *order* of g, denoted $order(g)$, is the smallest positive integer a such that $g^a = 1$.

Some basic facts that are all not too hard to prove and would be useful exercises:

- For every $g \in \mathbb{G}$, the map $a \mapsto g^a$ is a k to 1 map from $\{0, \ldots, |\mathbb{G}| - 1\}$ to $\mathbb{G}$ where $k = |\mathbb{G}|/order(g)$. See footnote for hint[7]
- As a corollary, the order of g is always a divisor of $|\mathbb{G}|$.
- An element g of $\mathbb{G}$ is called a *generator* if $order(g) = |\mathbb{G}|$. A group is called *cyclic* if it has a generator. If $\mathbb{G}$ is cyclic then there is a (not necessarily efficiently computable) *isomorphism* $\phi : \mathbb{G} \to \mathbb{Z}_{|\mathbb{G}|}$ which is a one-to-one and onto map satisfying $\phi(g \cdot h) = \phi(g) + \phi(h)$ for every $g, h \in \mathbb{G}$.

When using a group $\mathbb{G}$ for the Diffie Hellman protocol, we want the property that g is a *generator* of the group, which also means that the map $a \mapsto g^a$ is a one to one mapping from $\{0, \ldots, |\mathbb{G}| - 1\}$ to $\mathbb{G}$. This can be efficiently tested if we know the order of the group and its factorization, since it will occur if and only if $g^a \neq 1$ for every $a < |\mathbb{G}|$ (can you see why this holds?) and we know that if $g^a = 1$ then a must divide $\mathbb{G}$ (and this?). It is not hard to show that a random element $g \in \mathbb{G}$ will be a generator with non trivial probability (for

[7]You can show a one to one mapping between the set $\{a : g^a = 1\}$ and the set $\{b : g^b = f\}$ by choosing some element b from the latter set and looking at the map $a \mapsto a + b \pmod{|\mathbb{G}|}$.

similar reasons that a random number is prime with non trivial probability) and hence an approach to getting such a generator is to simply choose g at random and test that $g^a \neq 1$ for all of the fewer than $\log |\mathbb{G}|$ numbers that are obtained by taking $|\mathbb{G}|/q$ where q is a factor of $|\mathbb{G}|$.

Digital Signatures

Public key encryption solves the *confidentiality* problem but we still need to solve the *authenticity* or *integrity* problem, which might be even more important in practice. That is, suppose Alice wants to endorse a message m that *everyone* can verify but only she can sign. This of course is extremely widely used in many settings, including software updates, web pages, financial transactions, and more.

Definition: A triple of algorithm (G, S, V) is a chosen-message-attack secure *digital signature scheme* if it satisfies the following:

- On input 1^n, the probabilistic *key generation* algorithm G outputs a pair (s, v) of keys, where s is the private *signing key* and v is the public *verification* key.
- On input a message m and the signing key s, the signing algorithm S outputs a string $\sigma = S_s(m)$ such that with probability $1 - negl(n)$, $V_v(m, S_s(m)) = 1$.
- Every efficient aversary A wins with at most negligible probability the following game:
    1. The keys (s, v) are chosen by the key generation algorithm.
    2. The adversary gets the inputs 1^n, v, and black box access to the signing algorithm $S_s(\cdot)$.
    3. The adversary *wins* if they output a pair (m^*, σ^*) such that m^* was *not* queried before to the signing algorithm and $V_v(m^*, \sigma^*) = 1$.

The Digital Signature Algorithm (DSA)

The Diffie-Hellman protocol can be turned into a signature scheme. This was first done by ElGamal, and a variant of his scheme was developed by the NSA and standardized by NIST as the Digital Signature Algorithm (DSA) standard. When based on an elliptic curve this is known as ECDSA. The starting point is the following generic idea of how to turn an encryption scheme into an *identification protocol*. If Alice published a public encryption key e, then she can prove her identity to Bob by decrypting a random ciphertext c. However, this falls short of a signature scheme in two aspects:

- This is only an identification protocol and does not allow Alice to endorse a particular message m.

- This is an *interactive* protocol, and so Alice cannot generate a static signature based on m that can be verified by any party without further interaction.

The first issue is not so significant, since we can always have the ciphertext be an encryption of $H(m)$ where H is some hash function presumed to behave as a random oracle. The second issue is more serious. We could imagine Alice trying to run this protocol on her own by generating the ciphertext and then decrypting it, but this does not really prove that she knows the corresponding private key. After all, even without knowing d, any party can generate a ciphertext c and its corresponding decryption. The idea behind the DSA protocol is that we require Alice to generate a ciphertext c and its decryption satisfying some additional extra conditions, which would prove that Alice truly knew the secret key.

The algorithm works as follows: (See also Section 12.5.2 in the KL book)

- *Key generation:* Pick generator g for $\mathbb{G}$ and $a \in \{0, \dots, |\mathbb{G}| - 1\}$ and let $h = g^a$. Pick $H : \{0,1\}^\ell \to \mathbb{G}$ and $F : \mathbb{G} \to \mathbb{G}$ to be some functions that can be thought of as "hash functions".[8] The public key is (g, h) (as well as the functions H, F) and secret key is a.
- *Signature:* To sign a message m, pick b at random, and let $f = g^b$, and then let $s = b^{-1}[H(m) + a \cdot F(f)]$ where all computation is done modulo $|\mathbb{G}|$. The signature is (f, s).
- *Verification:* To verify a signature (f, s) on a message m, check that $s \neq 0$ and $f^s = g^{H(m)} h^{F(f)}$.

Very roughly speaking, the idea behind security is that on one hand s does not reveal information about b and a because this is "masked" by the "random" value $H(m)$. On the other hand, if an adversary is able to come up with valid signatures then at least if we treated H and F as oracles, then if the signature passes verification then (by taking log to the base of g) the answers x, y of these oracles will satisfy $bs = x + ay$ which means that sufficiently many such equations should be enough to recover the discrete log a.

Putting everything together - security in practice.

Let us discuss briefly how public key cryptography is used to secure web trafic through the SSL/TLS protocol that we all use when we use `https://` URLs. The security this achieve is quite amazing. No matter what wired or wireless network you are using, no matter what country you are in, as long as your device (e.g., phone/laptop/etc..) and the server you are talking to (e.g., Google, Amazon, Microsoft etc..) is functioning properly, you can communicate securely without any party in the middle able to either learn or modify the contents of

[8] As noted in the KL book, in the actual DSA protocol F is *not* a hash function but rather some very simple function that is still assumed to be "good enough" for security.

your interaction.[9]

In the web setting, therre are *servers* who have public keys, and *users* who generally don't have such keys. Ideally, as a user, you should already know the public keys of all the entities you communicate with e.g., `amazon.com`, `google.com`, etc.. etc.. . However, how are you going to learn those public keys? The traditional answer was that because they are *public* these keys are much easier to communicate and the servers could even post them as ads on the *New York Times*. Of course these days everyone reads the *Times* through `nytimes.com` and so this seems like a chicken-and-egg type of problem.

The solution goes back again to the quote of Archimedes of "Give me a fulcrum, and I shall move the world". The idea is that trust can be *transitive*. Suppose you have a Mac. Then you have already trusted Apple with quite a bit of your personal information, and so you might be fine if this Mac came pre-installed with the Apple public key which you trust to be authentic. Now, suppose that you want to communicate with `Amazon.com`. Now, *you* might not know the correct public key for Amazon, but *Apple* surely does. So Apple can supply Amazon with a signed message to the effect of

> *"I Apple certify that the public key of Amazon.com is 30 82 01 0a 02 82 01 01 00 94 9f 2e fd 07 63 33 53 b1 be e5 d4 21 9d 86 43 70 0e b5 7c 45 bb ab d1 ff 1f b1 48 7b a3 4f be c7 9d 0f 5c 0b f1 dc 13 15 b0 10 e3 e3 b6 21 0b 40 b0 a3 ca af cc bf 69 fb 99 b8 7b 22 32 bc 1b 17 72 5b e5 e5 77 2b bd 65 d0 03 00 10 e7 09 04 e5 f2 f5 36 e3 1b 0a 09 fd 4e 1b 5a 1e d7 da 3c 20 18 93 92 e3 a1 bd 0d 03 7c b6 4f 3a a4 e5 e5 ed 19 97 f1 dc ec 9e 9f 0a 5e 2c ae f1 3a e5 5a d4 ca f6 06 cf 24 37 34 d6 fa c4 4c 7e 0e 12 08 a5 c9 dc cd a0 84 89 35 1b ca c6 9e 3c 65 04 32 36 c7 21 07 f4 55 32 75 62 a6 b3 d6 ba e4 63 dc 01 3a 09 18 f5 c7 49 bc 36 37 52 60 23 c2 10 82 7a 60 ec 9d 21 a6 b4 da 44 d7 52 ac c4 2e 3d fe 89 93 d1 ba 7e dc 25 55 46 50 56 3e e0 f0 8e c3 0a aa 68 70 af ec 90 25 2b 56 f6 fb f7 49 15 60 50 c8 b4 c4 78 7a 6b 97 ec cd 27 2e 88 98 92 db 02 03 01 00 01"*

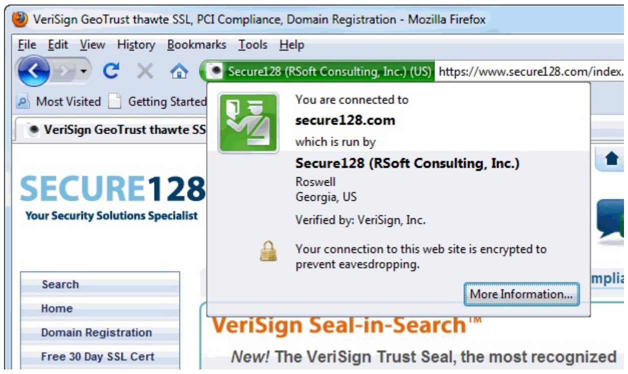
Such a message is known as a *certificate*, and it allows you to extend your trust in apple to a trust in Amazon. Now when your browser communicates with amazon, it can request this message, and if it is not present not continue with the interaction or at least display some warning. Clearly a person in the middle can stop this message from travelling and hence not allow the interaction to continue, but they cannot *spoof* the message and send a certificate for their own public key, unless they know Apple's secret key. (In today's actual implementation, for various business and other reasons, the trusted keys that come pre-installed in

[9] They are able to know that such an interaction took place and the amount of bits exchanged. Preventing these kind of attacks is more subtle and approaches for solutions are known as *steganography* and *anonymous routing*.

browsers and devices do not belong to Apple or Microsoft but rather to particular companies such as *Verisign* known as *certificate authorities.* The security of these certificate authorities' private key is crucial to the security of the whole protocol, and it has been attacked before.)

Using certificates, we can assume that Bob the user has the public verification key v of Alice the server. Now Alice can send Bob also a public *encryption* key e, which is authenticated by v and hence guaranteed to be correct.[10] Once Bob knows Alice's public key they are in business- he can use that to send an encryption of some private key k which they can then use for all the rest of their communication.

This is in a very high level the SSL/TLS protocol, but there are many details inside it including the exact security notions needed from the encryption, how the two parties negotiate *which* cryptographic algorithm to use, and more. All these issues can and have been used for attacks on this protocol. For two recent discussions see this blog post and this website.

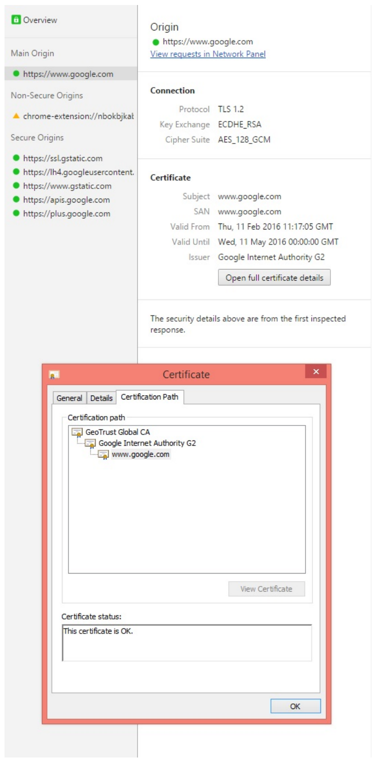

VeriSign GeoTrust thawte SSL, PCI Compliance, Domain Registration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Secure128 (RSoft Consulting, Inc.) (US)
https://www.secure128.com/index.
Most Visited
Getting Started
VeriSign GeoTrust thawte SS
SECURE128
Your Security Solutions Specialist
Search
Home
Domain Registration
Free 30 Day SSL Cert
More Information...
VeriSign Seal-in-Search™
New! The VeriSign Trust Seal, the most recognized

You are connected to
secure128.com
which is run by
Secure128 (RSoft Consulting, Inc.)
Roswell
Georgia, US
Verified by: VeriSign, Inc.
Your connection to this web site is encrypted to prevent eavesdropping.

Figure 1: When you connect to a webpage protected by SSL/TLS, the Browswer displays information on the certificate's authenticity

> **Example:** Here is the list of certificate authorities trusted by default by the Mozilla products: Actalis, Amazon, AS Sertifitseerimiskeskuse (SK), Atos, Autoridad de Certificacion Firmaprofesional, Buypass, CA Disig a.s., Camerfirma, CerticÃ¡mara S.A., Certigna, Certinomis, certSIGN, China Financial Certification Authority (CFCA), China Internet Network Information Center (CNNIC), Chunghwa

[10]If this key is *ephemeral*- generated on the spot for this interaction and deleted afterward- then this has the benefit of ensuring the *forward secrecy* property that even if some entity that is in the habit of recording all communication later finds out Alice's private verification key, then it still will not be able to decrypt the information. In applied crypto circles this property is somewhat misnamed as "perfect forward secrecy" and associated with the Diffie Hellman key exchange (or its elliptic curves variants), since in those protocols there is not much additional overhead for implementing it (see this blog post). The importance of forward security was emphasized by the discovery of the Heartbleed vulnerability (see this paper) that allowed via a buffer-overflow attack in OpenSSL to learn the private key of the server.


Overview

Main Origin

● https://www.google.com

Non-Secure Origins

chrome-extension://nbokbjkal

Secure Origins

● https://ssl.gstatic.com
● https://lh4.googleusercontent.
● https://www.gstatic.com
● https://apis.google.com
● https://plus.google.com

Origin

● https://www.google.com
View requests in Network Panel

Connection

Protocol	TLS 1.2
Key Exchange	ECDHE_RSA
Cipher Suite	AES_128_GCM

Certificate

Subject	www.google.com
SAN	www.google.com
Valid From	Thu, 11 Feb 2016 11:17:05 GMT
Valid Until	Wed, 11 May 2016 00:00:00 GMT
Issuer	Google Internet Authority G2

The security details above are from the first inspected response.

Certificate

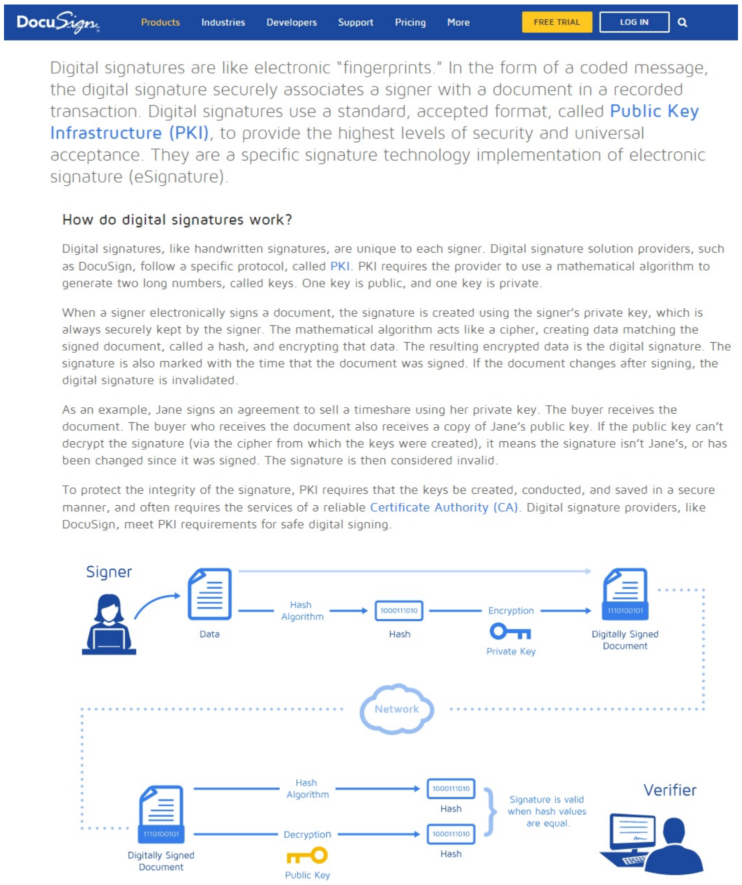
Certification path

```
GeoTrust Global CA
  Google Internet Authority G2
    www.google.com
```

Certificate status:

This certificate is OK.

Figure 2: The cipher and certificate used by ‘”Google.com”’. Note that Google has a 2048bit RSA signature key which it then uses to authenticate an elliptic curve based Diffie Hellman key exchange protocol to create session keys for the block cipher AES with 128 bit key in Calois Counter Mode.



Digital signatures are like electronic “fingerprints.” In the form of a coded message, the digital signature securely associates a signer with a document in a recorded transaction. Digital signatures use a standard, accepted format, called Public Key Infrastructure (PKI), to provide the highest levels of security and universal acceptance. They are a specific signature technology implementation of electronic signature (eSignature).

How do digital signatures work?

Digital signatures, like handwritten signatures, are unique to each signer. Digital signature solution providers, such as DocuSign, follow a specific protocol, called PKI. PKI requires the provider to use a mathematical algorithm to generate two long numbers, called keys. One key is public, and one key is private.

When a signer electronically signs a document, the signature is created using the signer’s private key, which is always securely kept by the signer. The mathematical algorithm acts like a cipher, creating data matching the signed document, called a hash, and encrypting that data. The resulting encrypted data is the digital signature. The signature is also marked with the time that the document was signed. If the document changes after signing, the digital signature is invalidated.

As an example, Jane signs an agreement to sell a timeshare using her private key. The buyer receives the document. The buyer who receives the document also receives a copy of Jane’s public key. If the public key can’t decrypt the signature (via the cipher from which the keys were created), it means the signature isn’t Jane’s, or has been changed since it was signed. The signature is then considered invalid.

To protect the integrity of the signature, PKI requires that the keys be created, conducted, and saved in a secure manner, and often requires the services of a reliable Certificate Authority (CA). Digital signature providers, like DocuSign, meet PKI requirements for safe digital signing.

Figure 3: Digital signatures and oher forms of electronic signatures are legally binding in many jurisdictions. This is some material from the website of the electronic signing company DocuSign

Telecom Corporation, Comodo, ComSign, Consorci AdministraciÃ³ Oberta de Catalunya (Consorci AOC, CATCert), Cybertrust Japan / JCSI, D-TRUST, Deutscher Sparkassen Verlag GmbH (S-TRUST, DSV-Gruppe), DigiCert, DocuSign (OpenTrust/Keynectis), e-tugra, EDICOM, Entrust, GlobalSign, GoDaddy, Government of France (ANSSI, DCSSI), Government of Hong Kong (SAR), Hongkong Post, Certizen, Government of Japan, Ministry of Internal Affairs and Communications, Government of Spain, Autoritat de CertificaciÃ³ de la Comunitat Valenciana (ACCV), Government of Taiwan, Government Root Certification Authority (GRCA), Government of The Netherlands, PKIoverheid, Government of Turkey, Kamu Sertifikasyon Merkezi (Kamu SM), HARICA, IdenTrust, Izenpe S.A., Microsec e-SzignÃ³ CA, NetLock Ltd., PROCERT, QuoVadis, RSA the Security Division of EMC, SECOM Trust Systems Co. Ltd., Start Commercial (StartCom) Ltd., Swisscom (Switzerland) Ltd, SwissSign AG, Symantec / GeoTrust, Symantec / Thawte, Symantec / VeriSign, T-Systems International GmbH (Deutsche Telekom), Taiwan-CA Inc. (TWCA), TeliaSonera, Trend Micro, Trustis, Trustwave, TurkTrust, Unizeto Certum, Visa, Web.com, Wells Fargo Bank N.A., WISeKey, WoSign CA Limited