Handshakes with Strangers

Security Protocol Design in Networked Applications

Alexander Horn

DePaul University SNL/CDM GIS

April 25 2011 – First Outline

Sept 2 2011 - Revised outline

Nov 2011 - First Draft

Dec 2011 – Revised

Jan 2011 – Revised x 2

Introduction & Background

Secrets are as old as language itself. Codes are often used to communicate secrets. However codes that worked for Sparta and Cesar didn't work well for Mary Queen of Scotts or German forces in WWII, as their secrets were revealed. Similarly, today, we see private codified communication made public - from Enron emails to Wikileaks to identity theft and London police monitoring all cell phone conversations in a 10km radius (Gallagher and Syal 2011. Modern day secrets seem impossible. Can anything still be hidden?

To answer this question, we will take an overview of the evolution of codes from the basic principles of cryptography through today's use in digital networks. We then explore problems with usage, possible improvements, as well as glimpse at future adaptations and concerns.

Cryptography

1.1 A brief history

People need to communicate secrets, yet written language has the chance of being seen by all. How to keep a secret? One way is to simply conceal the message. Hiding communications is known as Steganography. It's both commonplace (putting a letter inside an envelope) and unusual (writing on the inside of an egg for Renaissance diplomats, or swallowing silken wax balls for Chinese messengers (Singh 1999)) – yet concealing communication doesn't concern us in this paper. Rather, we explore 'open secrets' created with cryptography - a form of encoding whereas even when the message is found and in plain sight (as with anyone on the same computer network), it cannot (easily) be interpreted. The word is Greek, Kryptos – meaning Hidden (as opposted to Stegos, which means 'covered'). Where we find civilization, we find cryptography. Some of the earliest uses on record were Sparta, as documented in Herodotus' chronicles of Greece and Persian war in 500 BC (ibid). Secret writing was critical to Greek success against Xerxes (Ibid). Caesar used it to communicate with Cicero in the Gallic Wars

(ibid). Middle ages the Arabs continued the craft and discovered new ciphers. The first European cryptanalyst was Venetian Giovanni Soro in early 1500s (ibid). Yet cryptography is not just for diplomats and generals – personal use of cryptography was first evidenced in 400 BC within the Karma Sutra's advocacy of learning codes for personal note taking and communication, a sentiment later reflected with PGP in the 1990s (ibid).

Computing and cryptography have evolved together, as the speed of modern processors have allowed mathematics to be used that, while known, was previously impractical to implement. This same development also makes old codes increasingly easy to break. Furthermore, a limitation that has always plagued the history of cryptography remains: in order to decode each other's messages, parties must first exchange secret key(s). Those keys rely on a possibly compromised 3rd party 'out-of-band' communication, which adds complexity and vulnerability. Banks in the early 1970s employed teams of messengers to personally deliver the week's keys; not unlike German monthly updates of their Enigma code in WWII, or the literally tons of keys shipped out daily from US contractor COM-SEC in the1970s (Singh 1999). While computers allow new types of keys (public and private), and computer networks enable a new form of distribution, key management remains a problem today.

Furthermore, while the mathematical principles (integer factorization, discrete logarithm, elliptic curves) (Pfleeger and Pfleeger 2006) of today's cryptography are sound, every implementation seems to have vulnerabilities. But we're getting ahead of ourselves; let's back up a bit and examine the basics of cryptography.

1.2 Form and Function

With such a rich history and importance to daily life, there are actually very few principles at work. At the core of it all is a few algorithms – literally, two: Transposition, and Substitution (Singh

1999). These two fundamentals divide all of cryptography from antiquity through present day: Either method used by sender, followed in reverse by receiver, will result in the original message.

1.2.1 Transposition and Substitution

The transposition algorithm is as simple as re-arranging the letters, like an anagram. The first known military cryptography, the Spartan Scytale (dating back to 500 BC), used transposition (ibid). The message was written on a shoestring-like ribbon that revealed the message only when wrapped around a cone with the same dimensions as the sender. Without the same size cone, the message was unintelligible.

The Substitution cipher is the replacement of symbols with other symbols from a 'key'. The Substitution cipher first appears in 'mlecchita-vikalpha', Number 45 of the Kama Sutra's 64 courtesan arts (ibid). Militarily, it appears in Caesar's Gallic Wars – the cypher was as simple as replacing each letter with the letter three places further down (Ibid). For instance, with the algorithm 'shift one letter to the left', IBM becomes HAL. In other instances, Greek letters replaced Roman letters (Piper and Murphy 2002). Two thousand years later, the German Enigma machine form WWII was fundamentally a substitution cipher – albeit far more complex.

1.2.2 Evolution

With the dawn of computing, transposition and substitution alone were increasingly easy to decipher both with and without keys. Algorithms had to become more complex to offset the decreased cost of trial-and-error aka 'brute force' pattern matching. However, they still had substitution and transposition at heart, just with more steps.

The first post-war civilian cipher was Lucifer - a 48-bit 'block cipher'. A block cipher encodes a group of characters at once - as opposed to a stream cipher, which

encodes character-by-character[1]. Lucifer was developed by IBM in the 1970s and was used for banking, until the stronger Data Encryption Standard (DES) replaced it. DES worked very similarly, but with a 56 bit key. The size of the key becomes critical to modern cryptography; the length of time to brute force find the key increases roughly with the square of the key length (ibid). As computational power increases, what was once sufficient is no longer secure. For this reason DES was deemed insecure in mid 1990s and the industry moved on to AES. However, DES lives on in 'Triple DES' form, essentially DES with more cycles of transposition and substitution, and has yet to be broken and is deemed secure (Ibid).

Having to include ever-larger keys as well as alter algorithms is a symptom of ever-increasing computer power and the ongoing public & private effort to find flaws in cryptographic implementation. Yet this inevitability can be designed for. When DES was initially deployed in 1976, it was predicted to only be secure for about 20 years or so (ibid). So when DES was first broken in mid-90s, it was hardly a surprise and AES was ready to take its place.

1.2.3 **Asymmetric**

These ciphers are called symmetric; because one does the same thing the sender did, in reverse, in order to read it. However in 1976 a fundamentally new algorithm was conjured by Whitfield Diffie and Martin Hellman (Piper and Murphy 2002)(Singh 1999). Recall the catch-22 of all previous encryption schemes: before two people could exchange a secret message, they had to already have a shared secret key for encoding & decoding (Singh 1999). Using prime number factors as keys enabled each party to have a new kind of key, a key-pair, with both a public and private component. Someone may freely distribute their public key; anyone seeking to communicate securely with that person may use that key (combined with their private key) to encode the message, which is then decoded with

[1] Generally, stream ciphers are better suited to real-time communication (as in today's GSM phone networks), whereas block ciphers are better suited to communications with longer lifetimes (Piper and Murphy 2002).

the receiver's private key and the sender's public key (Pfleeger and Pfleeger 2006). This swapping of public and private keys for encoding and decoding is why it's called asymmetric – one encodes using a different method than one decodes, but still obtains the same result.

The primary innovation was this exchanging of a key, in the open, and still being able to ensure secrecy in two-party communication (Piper and Murphy 2002). RSA's impact on the field of cryptography cannot be underestimated (ibid).

This scheme grew into today's commonly used RSA algorithm, and PGP, among many others.

1.2.4 Exploits

All cryptography is based on mathematics. Whether the straightforward combinatorial mathematics of symmetric ciphers, or the more complex factorization of RSA or discreet logarithms of DSA, the theory behind the implementations indicate they're unbreakable. Enigma's stream cipher for instance had over 10^20 possible keys (Piper and Murphy 2002), which is more than some of today's ciphers – mathematically at the rate of one key a second this would take a billion times the lifetime of the universe to solve (Singh 1999). However it was cracked in time to shave two years off WWII (Piper and Murphy 2002).

All algorithms can be exploited in less than theoretical bounds due to flaws in implementation. WWII Germany's Enigma was exploiting both usage and key management mistakes. Similarly, today's encrypted hard drives can be decrypted because of known file contents; document types – PDF files contain the ASCII 'pdf' within the first few bytes, same for HTML and MS Word, Excel… when one knows even a little bit of the 'plaintext', cryptanalysis on encoded data can significantly accelerate breaking the keys (Piper and Murphy 2002). In a similar way, WEP wireless encryption can be broken in minutes due to the flawed stream cipher (RC4) implementation; as the same (short, pseudorandom) key has greater than 50 percent chance of being used twice

within 5000 packets (WEP 2011). Repeating a cipher and/or starting from the same point with a stream cipher is a bad implementation as it effectively 'leaks' key information into the encoded stream, easing cryptanalysis.

The only perfect cipher is Shannon's 'One-Time Pad', in which the key is not only as long as the message, but as long as all messages ever used (Piper and Murphy 2002). In this way the code can never be broken, as the cipher is always unique and never repeated. However the logistic limitations of the one-time pad (having to synchronize long keypads out-of-band) prevent it from being implemented, as distributing a key as long as one's message in a side-channel is impractical. This is known as the key distribution problem. Key management (including generation, distribution, storage, change, and destruction) remains one of the most difficult aspects of secure communication (ibid), even with the development of Asymmetric key pairs.

1.3 Algorithms in Use Today

Why do we use cryptography today? So far we've only discussed privacy – yet there are more subtle uses that are just as frequent. A primary challenge security professionals face is the development of good electronic substitutes for social mechanisms like envelopes, signatures, facial recognition, duplication, handshakes, letterheads, ID cards, and so forth (Piper and Murphy 2002). These uses can be captured by the terms *Authentication*, *Integrity*, and *Temporality* (also called *Availability*) (Forouzan 2006) (Piper and Murphy 2002) (Singh 1999).

Authentication means that the sender's identity is indeed the sender's, and not an imposter. Nonrepudiation is the ability of the receiver to prove this identity. The burden of proof is on the receiver (Forouzan 2006).

Integrity is the ability to ensure the message has not been altered in transmission. This is often solved with a Message Authentication Code (MAC), a collision-resistant hash function (similar to 'checksums') that gives a small (often 256 bit) deterministic fingerprint based on an input message (Cormen et al 2011). If the receiver's HMAC does not match the sent HMAC, the message has been altered.

Temporality is the ability to know when the trust was 'bound', and how long it is valid for. This is often solved by encoding a timestamp along with the message prior to encryption or hashing (ibid).

Digital Signatures (also called Certificates) are cryptographic techniques that encapsulate Authentication, Integrity, and Availability (Pfleeger and Pfleeger 2006). We find RSA (in x509 form) the most popular form of cryptographic certification in use today (ibid).

Together, these four uses of *privacy*, *authentication*, *integrity*, and *temporality* account for all use of cryptography in modern computing.

Protocol Analysis

Now that we have a sense of cryptographic mechanisms and their primary uses, we now take a detailed look at how encryption is used within popular modern protocols, so we may get a better sense of what problems have been solved, and what remain for future research and/or may remain entirely intractable.

1.4 SSL/TLS

Transport Layer Security (TLS) is SSL 1.2, and is one of the most common network encryption methods used today as it's used to ensure *privacy* and *integrity* of information exchanged between browsers and websites (Piper and Murphy 2002). We'll look at how it came about, how it works, and survey vulnerabilities and possible futures.

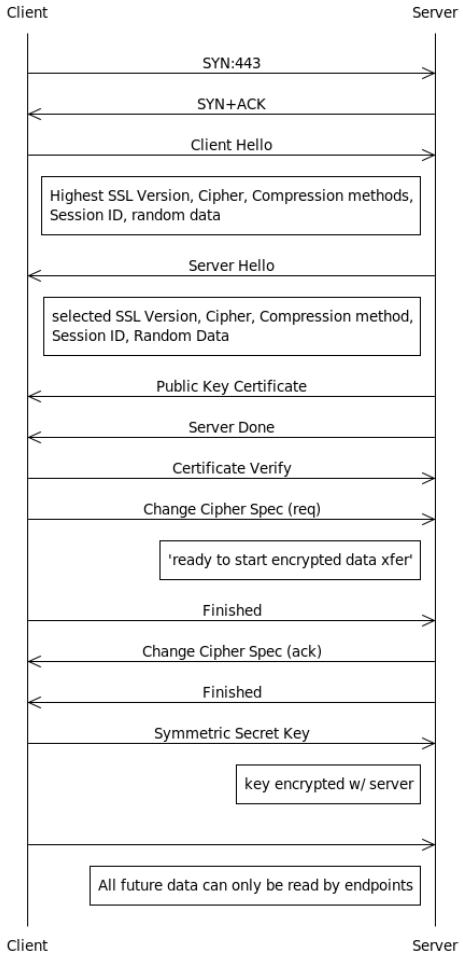
1.4.1 History

SSL was developed by Netscape to provide security for end-to-end communication on the World Wide Web (Forouzan 2006). Specifically, a browser needs authentication, Integrity, and privacy. For example, a user is to share their credit card - they want to know who they're sharing it with (*authentication*) and they don't want anyone to see it (*privacy*). Furthermore the vendor wants to ensure the message arrives intact; that quantity or price has not changed in transit (*integrity*), nor expired (*availability*).

1.4.2 Protocol Details

TLS is a non-proprietary form of SSL that has replaced SSL as it's been upgraded. The position of TLS in the Internet Layer is between application and transport – that is, between HTTP and TCP/IP. In the TCP/IP stack, TLS is a 'service' layer (ibid).
TLS itself has internal protocols – Handshake, Cipher spec, and Alert protocol (ibid). Handshake is used to set the session keys for client and server, cipher spec decides on the type of encryption algorithm (many are supported to allow backward compatibility). Alert is for error handling. Each protocol appends data into and controls the encoding of the application's message before inserting the TLS 'packet' into the TCP/IP layer (ibid).
A client-server protocol, the client is a browser and the server is a web host. The client initiates the connection. The handshake and cipher spec initiate the session, in a call and response that accomplishes the following (Forouzan 2006):

1. Two Parties agree on three protocols: entity authentication protocol, a message authentication protocol, and an encryption/decryption protocol.
2. The entity authentication protocol is used to authenticate two parties to each other and establish a secret between them
3. Each party uses a predefined function to create session keys and parameters for the message authentication protocol and encryption/decryption protocol
4. A hash digest is calculated and appended to each message to be exchanged using the message authentication protocol and the corresponding keys/parameters
5. The message and digest are encrypted using the encryption/decryption protocol and the corresponding keys/parameters
6. Each party extracts the necessary keys and parameters needed for the message authentication and encryption/decryption.



1.4.3 Vulnerabilities

Despite SSL's success – effectively driving all browser-based banking and secure log in – there are flaws in it's implementation. As we saw with cryptography in general, a cryptosystem's implementation is never as strong as the mathematics; this is quite evident in SSL/TLS.

TLS's primary flaw is it's Public Key Infrastructure – that is, key/certificate management (Elgamal 2011) (Forouzan 2006). As mentioned, key management is an unsolved area with ongoing research – yet addressing TLS vulnerability is important, as it's in-use by millions daily. Two aspects to key management specifically seem in need of improvement (Elgamal 2011). One is Certificate Authority (CA) compromise, and the other is the problem of in-browser static root of trust. They are related but distinct problems. Central to each is the 'outsourcing' of vendor/browser trust to CAs (Verisign, DigiNotar, etc) that are then responsible for initially verifying identity in more traditional means (photographs, business address, license, tax forms – thorough identity check, much like a bank underwriter) and issuing a digital certificate after doing so. That certificate then gets loaded onto the vendor's webserver, and is used to provide authentication for future HTTPS traffic. Yet what happens when the security of the CA itself is compromised? One can issue false certificates (Elgamal 2011). This is not theory; this happened during the 'Arab Spring' uprising in 2012, DigitNotar CA was compromised (allegedly by Iranian intelligence) and fake SSL certificates were issued for Twitter and online email – essentially allowing the government to spy on the citizens, removing all privacy without user's knowledge (ioerror 2011). Each browser vendor had to update their black list, some had to roll out a patch to upgrade the browser; which is our second vulnerability. Slow browser upgrades are the second largest flaw to TLS beyond Public Key Infrastructure (PKI). There are fixes to vulnerabilities that have yet to be implemented (Kaie 2011). This is no doubt due to the social complexity of doing so – there are 4 main

web browsers, each needing updating, yet to have fixes waiting idle for 3 years (ibid) seems inexcusable, especially when the consequences are so dire.

The third largest flaw is that TLS exists on top of TCP/IP, which is always vulnerable to man in the middle attacks. Such attacks are not easy, but are possible (taking advantage of fabricating the 'session counter' in TCP three-way-handshake in TLS step 2), and result in otherwise secure information being plaintext without either endpoint's knowledge. Similarly, there is never a 'pure' TLS connection – it's always instantiated from a 'plaintext' insecure connection (i.e.; the first 8 messages in sequence diagram), and that fact can be exploited on the client's machine (Wisniewski 2011). It's for this reason that spyware is dangerous; it can inject unnoticed patterns into browser URLs that can then serve as content analysis and allow a third party to decrypt information without user knowledge (Goodin Sept 2011). There are other flaws (such as spoofing leaf notes with wildcards in rarely-used portions of SSL certificate (Goodin Aug 2011)) but for the sake of this paper, we'll stop here.

What to do? While TLS cannot avoid TCP/IP, faster updates to known vulnerabilities coupled with user-management of root trust (verses hardcoding, requiring browser update) if not incorporating web-of-trust features (user-maintained organic trust gradients vs default static root trust) rather than keeping with existing PKI would aid in mitigation of CA compromise. Faking a root and it's tree is far easier than faking an entire forest. A recent new paper has been published by the industry to attempt to mitigate these issues (CA 2011).

1.5 Amazon EC2 Web Services best practice

Amazon's Elastic Compute Cloud (EC2) is a virtualized multi-tenant on-demand computer-as-utility service. The advantage to this cloud infrastructure is that one only pays for the resources one uses; so rather than investing in the infrastructure to handle one's peak traffic (i.e., seasonal, or weekly, or perhaps even random); one only needs to pay for the (elastic) infrastructure used (Amazon 2011). The caveat is that one's

applications must be architected with extreme care to keep data secure, as 'noisy neighbors' can arise with greater frequency and capacity for harm than on the Internet at large (Craig 2011). The goal of this protocol is *authentication* and *integrity* – yet as we'll see, minor changes can be made to allow for *availability* and *privacy*.

1.5.1 History

Combined with Elastic Load Balancer, Amazon's EC2 enables a web developer to (among other things) automatically scale a website to avoid Denial of Service (DOS). That is, in response to increased traffic, the system can increase resources (start new virtual machine instances, change routing & load balancing to handle them) to handle viral or sustained heavy loads, and then scale back down when no longer needed (Amazon 2011). Yet the load balancers have their own IP, and IPs are re-used; there's a chance you'll get another site's traffic (or someone else will get yours) during these scaling instance transitions (Craig 2011). Amazon sets IP Time-To-Live (TTL) extremely low on this internal traffic, so DNS routing through these new load balancer instances generally behaves acceptably – but sometimes there is spillover and tenant's traffic is shared (ibid). The result is that one cannot always trust one's own hostname or DNS as one would in a single tier environment such as your own non-cloud host. One incident had Netflix API traffic (2 million requests) access an (unrelated) user's account (Kalla 2011) due to delays in DNS propagation of the repurposed load balancer IPs.

As such when deploying an application on Amazon Web Services with Elastic Load Balancers, developers need to assume that random (untrusted) sources are reading client requests to the application's server (inbound traffic) (ibid). To account for this, Amazon suggests using a cryptographic hash at each endpoint to ensure each HTTPS request is indeed coming from the developer's intended application (*authentication*) (Craig 2011). This hash is done via HMAC (Hash Message Authentication Code) on the URL (with arguments) using a private key. In this way, an application can ignore any traffic that is not signed by your personal key, as the HMAC cannot be forged. As a side

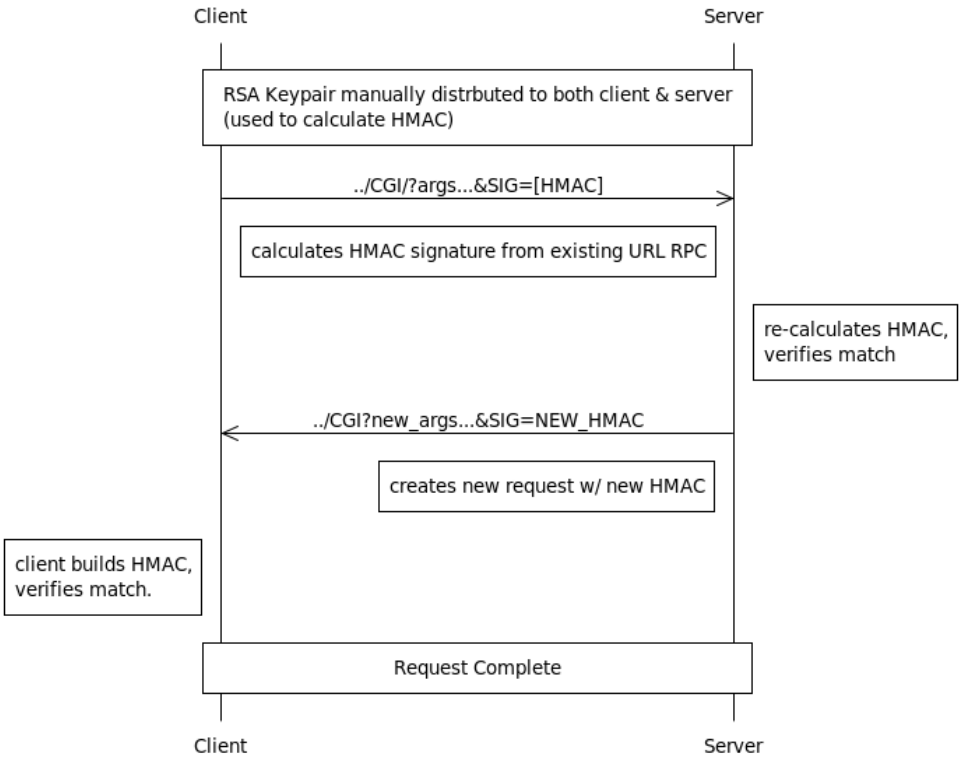
effect, integrity is also ensured; as the HMACs will not match if any of the URL parameters have been altered.

1.5.2 **Protocol Details**

1. Application Setup/Configuration

   server and client contain single RSA keypair. (only server and client know the private key)

2. Client creates HMAC representing entire request – entire URL and arguments.
3. Client sends HMAC as an argument, along with all initial arguments.
4. Server gets request and builds HMAC on received parameters.
5. Server creates own HMAC, and verifies it's HMAC == received HMAC
6. Server builds and sends request with corresponding new HMAC.
7. Client verifies HMAC match; request is valid and complete.



1.5.3 Vulnerabilities

The system depends on placing a private key in both client and server – which is possible when designing an application that has discrete control over all client/server endpoints, but not possible on more ad-hoc uses like SSL/TLS is designed for. Due to the small scale (few servers for application) the key distribution problem is a non-issue.

Additionally, this system runs on top of TLS, and so inherits TCP/IP pre-handshake plaintext vulnerabilities. One could use this same scheme over just HTTP – but then the user may notice the browsers 'insecurity' (lack of visual icon/padlock, no HTTPS in the location) and refuse to continue.

TLS difficulties aside, there are no Availability/Temporal bound (no time-out) limitations, so while the protocol may be immune to accidental reflection/DNS lag, it's still vulnerable to intentional replay attacks due to lack of application-specified sequence number.

In summary, this protocol could be improved easily by adding *temporality* and *privacy*. *Temporality* can be achieved by adding a state counter (time stamp w/ synchronized clocks or known UTC offset), and *privacy* can be used on required parameters by encrypting any critical elements in message prior to hashing.

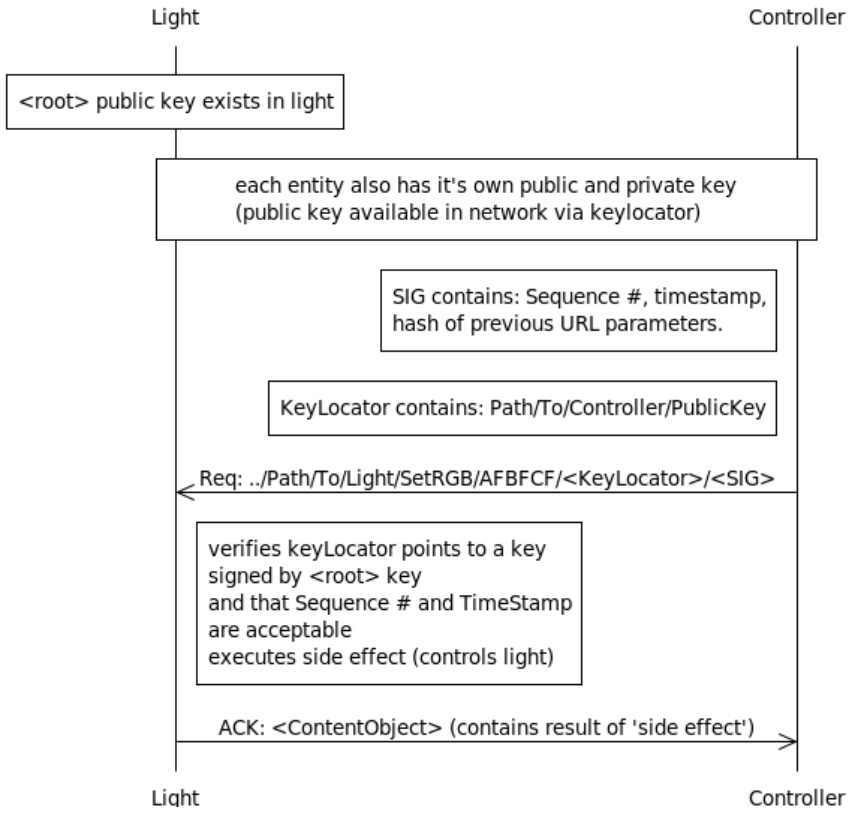
1.6 NDN 'Signed Interests'

1.6.1 History

NDN is nothing short of a redesign of the Internet with homage to TCP/IP. It currently runs as an overlay on top of TCP/IP, but could replace everything above the link layer, eliminating IP packets entirely when communicating between NDN nodes (Smetters and Jacobsen 2009). There are many distinguishing characteristics between NDN and TCP/IP, yet the only ones we concern ourselves with for this protocol are the fundamental difference in packets. Whereas IP has the datagram (single packet with sender & receiver address), NDN has two packets: the interest, and the content object. The Interest takes the place of an initial HTTP request, and the Content Object takes the place of the response.

Interests are completely anonymous (only pointing back to last hop, not sender end-point as in TCP/IP), Content Objects are cryptographically signed to ensure providence. This works well for HTTP-like content distribution. Yet what of SCADA-like control schemes (used in power plants, water purification, industrial manufacturing, uranium centrifuges, and so forth)? The nonexistent identity in the interest packet doesn't lend itself well to control; one does not want to execute anonymous control requests. Thus, how to issue commands to control a system? That is, how to design a packet that has *authentication*, *integrity*, *temporality*, and optional *privacy*? The answer again lies with public/private key pairs, as well as Hashed Message Authentication Codes (HMAC).

1.6.2 Protocol Details



Command certification is accomplished within NDN by combining two features: a repurposed 'KeyLocator' from the NDN Content Object, and a cryptographic component that consists of a Time Stamp, a counter, as well as the 'name' (like an HTTP CGI URI).

The cryptographic component can be signed and verified using either symmetric or asymmetric encryption – symmetric is faster, thus desirable for control once symmetric is used to establish trust (especially on low-power embedded processors).

Repudiation is possible from the controller through the 'KeyLocator' object, which resolves to a verified public key.

Generally, rather than rigid hierarchical PKI as we saw with SSL, NDN enables an application to design any trust scheme, simply helping in key distribution (as a content object). To do so, a variant of Simple Distributed Security Infrastructure (SDSI) is used – which is in it's loosest formation a web-of-trust where each entity builds a 'reputation' through referrals in the form of certificates - keys signing other parties' keys. Again, one can (possibly) fake a tree, but an entire forest is quite unfeasible.

1.6.3 **Vulnerabilities**

Currently 'state' is a single sequence counter per-public key (i.e., per application). This is fine when there is only one application communicating with the controller; but it will not scale for multiple applications in the same namespace as control messages will not verify deterministically if there are multiple applications issuing control commands. To fix this, one must increment a state per key, per RPC namespace. This is a bit like a separate TLS connection per command, per application.

Furthermore there is a shared 'root' – same problem we saw in TLS, with hardcoded root trust keys. Better use of SDSI to establish root trust should be explored, perhaps augmented by aggressive nonrepudiation.

And if any part of the command is sensitive, it can be encrypted before cryptographic hashing.

In this way, *availability*, *integrity*, *privacy*, and *authentication* can all be met in a suitable manner for NDN-based control applications.

Conclusions

1.7 Meta-patterns apparent?

1.7.1 Central Authority Vs Trust Gradients

Any central root of trust creates a high value target – compromise that trust, and one compromises the network. We saw that with TLS PKI, and existing NDN signed interest implementation. Web of trust and/or SDSI is an alternative, a better analog to our offline social trust networks, using public key signatures to serve as ‘reputation’ discoverable by crawling network in a sort of open-ended nonrepudiation algorithm. While interesting, this ‘web of trust’ has yet to be implemented in any real way except in experimental schemes like Bit Coin (and it’s implementation has proven vulnerable) (Yang 2011). In general, while asymmetric encryption has transformed key distribution, key management remains an open problem for ad-hoc open networks with many users.

1.7.2 As low in protocol layers as possible

Even if an application itself is perfectly secure – if it runs on top of TCP/IP, it is potentially vulnerable. Application (and/or service) encryption is not end-to-end; there are exploits within lower layers that can be used to decrypt application messaging. While somewhat infeasible in practice, the fact is TCP/IP is always vulnerable to man-in-the-middle (MITM); one can always get in the middle of a TCP handshake and hijack the session, therein getting any session keys for any higher-level protocols. There's no way around that without avoiding TCP/IP, which is generally a practical impossibility. However, there are emerging alternatives, NDN being the most general purpose and similar to IP (in that it can run over anything, including IP) yet has encryption at a packet level, through it's thin-waist. In NDN, applications can be truly end-to-end in their security. Yet while NDN seeks to address many of TCP/IP's vulnerabilities, new exploits undoubtedly exist and have yet to be discovered.

Regardless of transport layer, one must always be aware of the possibility of externalities and design for them (with best practices, HMAC, counters, custom PKI) as much as possible.

1.7.3 Contain Compromises in Time

In general, given that perfect security is impossible, the most relevant question for security is not 'is this an exceptionally secure system' but rather 'is this secure enough for the application' (Piper and Murphy, 2002). One common way of trying to determine level of security requires is to try to estimate the length of time for which the information needs protection – called 'cover time' (ibid). The (mathematically derived) estimated time required for an exhaustive key search should be significantly longer than the cover time.

1.8 Musings

1.8.1 Quantum computing

Quantum computing is said to render existing prime factoring and elliptical methods obsolete (Wood 2011). However at best it reduces factoring time by square root (ibid) – considerable reduction, but far from instantaneous. Furthermore there are different types of quantum computers. The only one currently on the market, the Adiabatic quantum computer, is of no use for encryption – it's only good at finding lowest-energy state of a Hamiltonian system (Rosen 2007), and encryption is not a Hamiltonian problem. Thus a quantum computer that will affect encryption is still just beyond the horizon.

Meanwhile of relevance to solving key distribution, is quantum key distribution (QKD). QKD uses the physical property of quantum entanglement to distribute authenticated inviolable keys and is physically impossible to eavesdrop on without destroying the message (Wood 2011).

1.8.2 Usability in Security

If a system is not easy to use, it will not be used correctly. Indeed, as we saw with TLS certificate wildcards (Goodin Aug 2011), even if a trust model is easy to use, it will not be used correctly! Any confusion in design can only make things worse. Usability and Human Computer Interaction is a familiar topic, but more investigation is required for usability's insight into security (Cranor and Garfinkel 2005).

1.8.3 Ethics and Epistemological regress

The NDN project recently had a social scientist and an attorney come on board as a 'Values in Design' to explore the social dimension of architectural decisions. Attempts at realizing social affordances of new technology should be applauded - for instance, bad PKI in TLS allows enhanced governmental control of a free society via ease of mass surveillance – and same for poor stream cipher in GSM. Like medicine and healthcare, there is a social dimension to software that should not be ignored.

So have we answered our original question: can secrets still exist, today? A viable answer may be: yes – for a limited (estimable) time.

In summary and conclusion, just as scientific (or, really, any) truth is a process of finding best approximation, any trust in encryption is relative. This notion of ‘fallibilism’ was advocated by Popper, Dewey, and others as a way out of epistemological regress: that we must try to get as close to truth as possible, and accept that as good enough. This fallibilism fits within security (nothing is every fully trusted, thus trust gradients and ‘cover time’) as well as AGILE software design (nothing is ever perfect, so release early & often). Note this reasoning is not an excuse to take shortcuts, but rather a reminder that no implementation can ever be perfect; we can only do our best and stay alert.

References

Amazon. Amazon Web Services Documentation. Amazon.com Web. 27 Nov 2011. http://aws.amazon.com/documentation/

Craig, AWS, ed. "Making Secure Requests to Amazon Web Services." Amazon Web Services. Amazon.com, July 6 2009. Web. 27 Nov 2011. http://aws.amazon.com/articles/1928.

Cranor, Lorrie Faith, and Simson Garfinkel. Security And Usability, Designing Secure Systems That People Can Use. O'Reilly Media, Inc., 2005.

Cormen, Thomas H, Charles E Leiserson, and Ronald L Rivest. Introduction to algorithms. Cambridge, Mass: MIT Press, 2009.

CA Browser Forum. Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v1.0. The CA Browser Forum. Dec 15 2011. Web. <http://cabforum.org/Baseline_Requirements_V1.pdf>

Elgamal, Taher. "Father of SSL says [it] has lots of life left." Network World. Interview by Tim Greene. Oct 11 2011. Web. <http://goo.gl/giX0g>.

Forouzan, B. (2006). *Tcp/Ip Protocol Suite*. New York, NY: McGraw-Hill.

Gallagher, Ryan and Rajeev Syal. Met Police using surveillance system to monitor mobile phones. Web. http://www.guardian.co.uk/uk/2011/oct/30/metropolitan-police-mobile-phone-surveillance. Retrieved Oct 31 2011.

Goodin, Dan. "Wildcard certificate spoofs web authentication." The Register. The Register, 30 July 2009. Web. 12 Aug 2011. <http://www.theregister.co.uk/2009/07/30/universal_ssl_certificate/>.

Goodin, Dan. "Hackers break SSL encryption used by millions of sites." The Register. The Register, 19 Sep 2011. Web. 20 Sep 2011. <http://www.theregister.co.uk/2011/09/19/beast_exploits_paypal_ssl/>.

IETF. TLS 1.2 Specification. IETF Web. Aug 2008. http://www.ietf.org/rfc/rfc5246.txt

Ioerror. The DigiNotar Debacle, and what you should do about it. Tor Project. Aug 31 2011. Web. Retrieved Sept 2 2011. https://blog.torproject.org/blog/diginotar-debacle-and-what-you-should-do-about-it

Kaie, Kuix. "Potential Vulnerability Status of Major Ecommerce Sites."SSLTLS.de. SSLTLS.de, 27 Nov 2011. Web. 27 Nov 2011. <https://ssltls.de/status-of-major-sites.php>.

Kalla, Riyad. "AWS Elastic Load Balancer sends 2 Million Netflix API requests to Wrong Customer."The Buzz Media. Buzz Media, 29 Oct 2011. Web. 2 Nov 2011. <http://www.thebuzzmedia.com/aws-elastic-load-balancer-sends-2-million-netflix-api-requests-to-wrong-customer/>.

Pfleeger, Charles, and Shari Pfleeger. *Security in Computing*. 4th. Upper Saddle River, NJ: Prentice Hall, 2006. Print.

Singh, Simon. *The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography*. 1st. New York, NY: First Anchor Books, 1999. Print.

Piper, Fred, and Sean Murphy. *Cryptography*. 1st. Oxford UK: Oxford Press, 2002. Print.

Rosen, Geordie. Adiabatic Quantum Explanation. D-Wave Systems. May 2007. Online http://www.youtube.com/watch?v=6BxlolQdI3s

Smetters, D. K.; Jacobson, V. Securing network content. PARC TR-2009-1; 2009 October.

"WEP (algorithm)." Wikipedia. Wikipedia, 24 Nov 2011. Web. 11 Aug 2011. <http://en.wikipedia.org/wiki/Wired_Equivalent_Privacy>.

Wisniewski, Chester. "EFF uncovers further evidence of SSL CA bad behavior." Naked Security. Sophos.com, 6 Apr 2011. Web. 7 Apr 2011. <http://nakedsecurity.sophos.com/2011/04/06/eff-uncovers-further-evidence-of-ssl-ca-bad-behavior/>.

Wisniewski, Chester. "Defcon 2011: SSL and the future of authenticity." Naked Security. Sophos.com, Aug 16 2011. Web. 20 Aug 2011. <http://nakedsecurity.sophos.com/2011/08/16/defcon-2011-ssl-and-the-future-of-authenticity/>.

Wood, Lamont. "The Clock is Ticking for Encryption." Computerworld. Computerworld, 21 Mar 2011. Web. 22 Mar 2011. http://www.computerworld.com/s/article/354997/The_Clock_Is_Ticking_for_Encryption

Yang, Edward Z. “Bitcoin is not decentralized”. June 1, 2011. Web June 7 2011. http://blog.ezyang.com/2011/06/bitcoin-is-not-decentralized/