Keys Don't Grow in Threes

Stephen Farrell • *Trinity College Dublin*

Many Internet security mechanisms depend on the use of cryptographic algorithms for various forms of authentication and confidentiality. Even when well-known and standardized cryptographic algorithms are used in well-known protocols, some parameters must be specified, the most important of which are usually algorithm identifiers and key or hash-output lengths. Here, I review some recent key length recommendations and compare those to current usage.

Strength Is Not Quite Length

Because this is an area where I've had to explain things to nonsecurity folks several times over the years, I'll start with a quick explanation of what we mean when we talk about key lengths. (Actually, this is also a useful way to spot people who don't know what they're talking about — if someone's marketing materials make great claims as to why they're better because they use 40,000-bit keys, then don't buy that product!)

Many Internet security mechanisms at some level use cryptography. Take, for example, accessing a secure Web site via an "https://" URL, which involves running the HTTP protocol over the Transport Layer Security (TLS) protocol (also known as Secure Sockets Layer [SSL]). This typically uses the Rivest, Shamir, and Adleman (RSA) asymmetric cryptographic algorithm to check the signature on the Web server certificate and to exchange a session key. In fact, that part of the protocol also requires a digest algorithm, such as the secure hash algorithm (SHA-1). After the key exchange, the browser and Web server use a symmetric algorithm such as the Advanced Encryption Standard (AES) to protect the information exchanged between the browser and the Web server. In this protocol as well as others, we often talk about using a suite of algorithms and sometimes call that ensemble a ciphersuite.

For each of these cryptographic uses, we should be concerned about the cryptosystem's strength. Not only are there different algorithms that can be configured (for example, Diffie-Helmann key agreement is an alternative to RSA), many of the algorithms they use can support different key lengths, which have different strengths.

By strength here, I mean something that indicates the amount of work that a knowledgeable and well-funded adversary would have to do to break the algorithm's security, as used in the protocol in question. So, for example, if I could quickly factor any number that's roughly 2,100 bits long, then I could masquerade as almost any Internet bank using RSA. Thankfully, we don't think this is practical for any adversary at this time.

These algorithms generally require a cryptographic key as input, or produce a certain size of output, and were those inputs or outputs very short, then the adversary probably wouldn't have much work to do at all. For example, if an algorithm can only accept 40-bit keys, then the system has only 1,099,511,627,776 (2 to the power of 40) keys in total, which, although a large number, isn't large enough: an adversary could simply try all possible keys until one works — a so-called "brute force" attack. Similarly, if a digest operation's output were small, say only 32 bits long, then an adversary could simply guess possible inputs until it found a matching output, probably after only roughly 65,000 attempts.

For most cryptographic algorithms, we can approximate the amount of work for the adversary via a single number, often directly related to the key length, or the output length for a hash algorithm. That's a good thing — it lets us compare different sets of parameters for the same algorithm and also ensure that we use sensible algorithm combinations — that is, sets of algorithms with commensurate strengths.

cont. on p. 94

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on February 23,2010 at 11:05:51 EST from IEEE Xplore. Restrictions apply.

cont. from p. 96

It's important to use commensurate strengths because an adversary always attacks the weakest link. There's a corollary to that — if you increase the strength for only one algorithm, the attacker will simply attack elsewhere, and because increasing strength generally requires more computation, we don't want to do that without benefitting from it.

However, you must understand that these numbers take into account only currently known attacks against the algorithms. For example, the SHA-1 algorithm has a 160-bit output and was designed to have 80 bits of strength against collisions. However, recent attacks[1] have demonstrated that, in fact, the adversary is really only up against roughly 63 bits of strength.

Similarly, different algorithms, say RSA and AES, have radically different reasonable key lengths associated with them because their internal designs are radically different. For example, a 128-bit AES key is considered strong today, whereas a 128-bit RSA key would be trivial to break — for RSA, we should be somewhere around 2,048 bits of key length, which is roughly equivalent to 112 bits of strength when compared to the AES algorithm. So, it's not entirely trivial to translate key length into algorithm strength, but a well-known relationship exists for all useful algorithms.

Why Worry?

Well, we've already seen one thing that system designers should consider — that is, the use of commensurate strengths for all the algorithms in a ciphersuite. However, we also need to bear in mind that all reasonable cryptographic algorithms can, in principle, be broken — just try all the keys in turn until you find the right one (the brute-force attack I mentioned earlier). If we increase the key strength and length, then we generally increase the work for the adversary, so longer keys equate to stronger ciphersuites. For most algorithms, we'd expect that each additional bit would essentially double the adversary's work because each additional bit doubles the number of possible keys. Of course, variations for different algorithms exist, but the basic idea that longer is stronger is, other things being equal, correct.

However, we don't usually increase key lengths by one bit at a time. First, having key lengths map well to various boundaries is often beneficial — for example, AES keys can be 128, 192, or 256 bits long but not 131 bits long due to the algorithm's internal design. For other algorithms, including RSA, the output size also depends on the key length. In fact, there are some so-called side-channel attacks in which unusual output lengths that don't fit on byte boundaries would make the attack easier. So, we prefer to deal with well-known lengths, such as 2,048 or 3,072 and not, for example, 2,051. (So, we don't grow keys in threes.) Basically, we're not dealing with a continuum here but with a fairly small range of well-analyzed lengths and strengths for each algorithm in our ciphersuites.

Because we don't want to allow practical brute-force attacks, we must estimate a reasonable set of strengths for each algorithm in our ciphersuites. But reasonable lengths change over time, as Moore's law increases the amount of work that the adversary can do for each unit of currency.

We can make some predictions based on Moore's law, but we generally can't predict advances in cryptanalysis (the art or science of breaking this kind of thing). For some algorithms, experts might be willing to recommend a change due to their expectation of future developments, based on the current state of the art. In this context, we also need to bear in mind that attacks never get worse — they only improve over time — so if an adversary has attacked some algorithm at some strength in some algorithm-specific manner, then we could guess that similar attacks will have some predictable effect on other algorithm strengths.

The main point is that, as time goes on, what was once a reasonable ciphersuite could become insecure. And if we trade off performance or bandwidth for strength, as we usually do in any real system, it's quite likely that we'll have to increase the strengths used in a matter of years, not decades.

Are We There Yet?

So, who's making recommendations, and what are they? Well, several cryptographers have tackled aspects of the problem, but various governments (including the US, Germany, and France) have also made more systematic recommendations. Damian Giry maintains a nice Web site (www.keylength.com) that summarizes and provides references for the various recommendations in an easy-to-compare form — this is helpful, given that commensurate strength is a goal here. The US National Institute of Standards and Technology (NIST) recommendations are perhaps the best known and date from 2007; they suggest, for example, that by the end of 2010, deployments should move to 2,048-bit RSA keys, whereas the current most common key size in use is a 1,024-bit RSA. In addition, NIST recommends moving to 112-bit strength for symmetric keys, though it's probably better to move to 128-bit AES keys. For hash algorithms, the position is perhaps a little different because the required strength depends on the purpose for which the hash function is used. In most cases, using SHA-256 (with a 256-bit output and, hopefully, roughly 128 bits of collision-resistance strength) is probably the best choice.

So, we have recommendations since 2007, and we have a set of

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on February 23,2010 at 11:05:51 EST from IEEE Xplore. Restrictions apply.

widely supported algorithms that match those recommendations. You might therefore ask how well deployments match those recommendations. In 2005 and 2006, Homin Lee and his colleagues[2] conducted a survey in which they probed roughly 19,000 Web servers supporting SSL or TLS and then logged the range of algorithms available, and preferred, for use with those servers. Perhaps the most notable finding there was that 88 percent of the servers they polled were using a 1,024-bit RSA with only 6 percent using a 2,048-bit RSA. Oddly, and perhaps for marketing reasons, 55 percent of servers supported AES-256, even though its design strength is vastly higher than the other cryptosystems in use. The versions of SSL and TLS tested didn't fully support different hash functions, so this survey didn't produce interesting results in that respect.

Updating

It therefore looks like we have some work to do to update the set of ciphers used on the Internet, and, if we follow the recommendations, we should start in 2009 for deployment in 2010.

Probably the first practical thing to do is to make an inventory of the use of cryptographic algorithms in your deployed environment. Although this sounds easy, in many real deployments, it could be time-consuming and messy. For example, various Web services might use load-sharing devices that actually terminate TLS connections, and it could be hard to inventory the set of services that use that TLS termination point. Similarly, virtual private network gateway devices might not easily list the set of algorithms they support — or getting that list might require work — and privileged access to the device in question could be difficult to obtain. System administrators could also use various network logging and analysis tools to determine which algorithms their systems actually use. Of course, depending on the protocol concerned, that might not be the full story — for example, if some aspect of algorithm negotiation actually occurs as part of an encrypted session. However, the fact that all protocols layered on top of TLS and IPsec use certificates at least makes the deployment of longer RSA keys and new certificate-signing algorithms (for example, RSA with SHA2) relatively easy. However, the equipment concerned must support the relevant algorithms.

With SSH keys, on the other hand, the situation is a little harder. Part of the beauty of SSH is the ability to take the leap of faith independently of other infrastructure, so that the administrator of a single server (or desktop) can enforce secure access without much overhead. That, however, can cause issues when it comes time to replace multiple keys or algorithms in that many hosts might need to be manually updated before the job is done.

The SSH update case also highlights a problem that's common when changing cryptographic parameters or algorithms. If one side makes a change (for example, generating a new, longer SSH RSA server key), but the other is unaware of that change, this could result in undesirable effects at the user interface on the unchanged side. With SSH, this occurs when a user attempts to log in to a previously known sever that has changed its key and results in the display of a somewhat scary warning message because the client can't distinguish a valid server key change from a man-in-the-middle (MITM) attack. If a site makes piecemeal changes to such server keys, then it's essentially training users to ignore those warnings, which can undermine the system's overall security in the event of a real MITM attack. Avoiding this problem might call for educating local users, which is, in any case, worthwhile.

It's also worth considering the possible effects of human error if administrators make wholesale changes to the cryptographic algorithms. Even with certificate-based systems — and certainly with manual systems such as SSH — administrators might make configuration errors when changing keys or selecting algorithms, and such errors could result in opening up an otherwise well-protected system when using the older settings.

The overall conclusion we can draw is that it's worthwhile for system and application administrators to take an inventory of their use of cryptography (or, more generally, of deployed security measures) and to maintain that inventory so that they can plan changes well in advance (or even in a hurry if the cryptographic sky does fall). This way, they can test the changes and circulate relevant information to users in advance of the actual changes. And, of course, administrators should plan to include updates to their cryptographic infrastructure as part of normal release cycles, taking into account the recommendations that the cryptographic community has developed.

References

1. X. Wang et al., "Finding Collisions in the Full SHA-1," *Advances in Cryptology — Crypto 2005*, LNCS 3621, Springer, 2005, pp. 17–36.
2. H.K. Lee et al., "Cryptographic Strength of SSL/TLS Servers: Current and Recent Practices," *Proc. 7th ACM Sigcomm Conf. Internet Measurement*, ACM Press, 2007, pp. 83–92.

Stephen Farrell is a research fellow at Trinity College Dublin. His research interests include security and delay/disruption tolerant networking. Farrell has a PhD in computer science from Trinity College Dublin. Contact him at stephen.farrell@cs.tcd.ie.

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on February 23,2010 at 11:05:51 EST from IEEE Xplore. Restrictions apply.