Deploying a New Hash Algorithm

Steven M. Bellovin
smb@cs.columbia.edu
Columbia University

Eric K. Rescorla
ekr@networkresonance.com
Network Resonance

Abstract

The strength of hash functions such as MD5 and SHA-1 has been called into question as a result of recent discoveries. Regardless of whether or not it is necessary to move away from those now, it is clear that it will be necessary to do so in the not-too-distant future. This poses a number of challenges, especially for certificate-based protocols. We analyze a number of protocols, including S/MIME and TLS. All require protocol or implementation changes. We explain the necessary changes, show how the conversion can be done, and list what measures should be taken immediately.

1 Introduction

Nearly all major cryptographic protocols depend on the security of hash functions. However, this is increasingly looking like a brittle foundation: although a variety of hash functions are available, only MD5 [32] and SHA-1 [23] are in wide use. Both hash functions derive from MD4 [31], which has long been known to be weak [7, 8], thus leading to concerns that they might have common weaknesses.

These concerns were borne out in late 2004, when techniques for efficiently finding collisions in MD5 [37] and SHA-0 [2] were announced. Subsequently, Wang [36] announced a technique for finding collisions in SHA-1 in 2^{69} operations,[1] rather than the 2^{80} for which it was designed, and Lenstra et al. [21] demonstrated a pair of X.509 certificates with the same distinguished name, different public keys, and identical signatures, though no extension is known which can generate such a pair with different distinguished names.

It should be emphasized at this point that none of these results have translated into demonstrable attacks on real-world protocols, though [21] comes uncomfortably close. However, it is clear that neither MD5 nor SHA-1 is as strong as its target security level and so need to be replaced. The possibility of new attacks lends some urgency to this transition.

[1]In a presentation delivered at the Rump Session of CRYPTO 2005, Shamir stated that Wang had improved the attack to 2^{63} operations. Also see `http://www.csrc.nist.gov/pki/HashWorkshop/2005/Oct31_Presentations/Wang_SHA1-New-Result.pdf`, the slides from Wang's keynote speech at the NIST Cryptographic Hash Workshop.

Although we don't discuss the issue in detail, most of our analysis applies to deploying new signature algorithms as well as to deploying new hash functions. If the signature algorithm is linked to a particular hash function, as DSA is tied to SHA-1, the two would change together; beyond that, since signature algorithms are almost always applied to the output of hash functions, if there is no easy way to substitute a new hash algorithm there is almost certainly no way to substitute a new signature algorithm, either.

2 Overview of Recent Hash Function Attacks

Conventionally, hash functions are designed to have three properties:

Collision resistance It is computationally infeasible to find $x, y, x \neq y$ such that $H(x) = H(y)$.

Preimage resistance Given an output value y, it is computationally infeasible to find x such that $H(x) = y$.

Second preimage resistance Given an input x', it is computationally infeasible to find x such that $H(x) = H(x')$.

The current generation of attacks address collision resistance. MD5 is effectively dead from that perspective; SHA-1 is much weaker than it should be, though finding collisions is still impractical.

While not as devastating as failures of the other two properties, collision resistance is indeed a serious issue. Lucks and Daum have generated Postscript files that exploit the attack (see `http://th.informatik.uni-mannheim.de/people/lucks/HashCollisions`). They took advantage of a well-known property of Merkle-Damgård hash functions:

$$H(x) = H(y) \Rightarrow H(x||\Sigma) = H(y||\Sigma)$$

where Σ is an arbitrary string, provided that x and y are the same length.

First, they generated two Postscript prologues that contained a collision in what was, syntactically, a constant. This constant was assigned to a variable. To each of these files, they then appended a Postscript program that checked the value of this variable and displayed one of two letters. An attacker could persuade someone to digitally sign the first, harmless letter; this same signature would match the second, harmful letter. Note, however, that to a great degree this attack is enabled by the fact that users do not directly view the Postscript code and rather use an interpreter. Similar attacks can be demonstrated against such systems (e.g., HTML with JavaScript) even without the ability to find hash collisions [30] by exploiting conditional elements in the display system [15, 16].

Collision-finding attacks do not rule out all uses of a hash function. In particular, the pseudo-random function properties are not affected at all. Furthermore, HMAC [20] is probably safe, since the unknown component—the key—of the inner hash function makes it impossible to generate a collision at that stage; this in turn helps protect the outer hash.

On the other hand, there is grave danger for many situations involving digital signatures or fingerprinting. If a would-be attacker can supply the message to be signed, that same attacker could have prepared two versions of the message, one innocuous and one harmful, while presenting only the former. The attacks work because the victim inspects the innocuous version and verifies that it is acceptable. In environments where victims do not carefully inspect data before it is hashed, collision attacks only modestly increase the threat level.

3 Overview of the Hash Transition Problem

Although the details of transition strategies for any given protocol may vary, there are many common elements. In this section, we provide an overview of the hash transition problem and the design goals that transition strategies should attempt to fulfill.

The hash transition problem is a special case of the general protocol transition problem. Whenever a new version of a protocol is rolled out, designers and implementors must figure out how to accomplish a smooth transition from old to new versions with a minimum level of disruption. In a typical protocol transitional environment, there are three types of agent:

Old Agents which only speak the older version.

Switch-hitting Agents which can speak both versions.

New Agents which can only speak the new version.

At the beginning of the transition, all agents are Old. At the end of the transition (at least theoretically), all agents are New. (In practice, of course, transitions of this nature tend to persist for an arbitrarily long time, since old systems never quite die off.) The purpose of a transition strategy is to accomplish the transition between these states with a minimum of disruption. A number of issues are common to most, if not all, such transitions:

Backward compatibility Old agents and Switch-hitting agents should be able to communicate using the older version. Without backward compatibility, users have an enormous disincentive to upgrade their implementations.

Newest common version When two Switch-hitting clients communicate, they can either use the new or old versions of the protocol. Because the purpose of the transition is to deploy the newer version, it is desirable to use that version where possible.

Downgrade protection An additional requirement for security protocols is to defend against version/algorithm downgrade. Consider the situation where two peers each support two cryptographic algorithms, one of which is strong and one of which is weak. If an attacker can force the peers to use the weaker algorithm, he may be able to attack the communication. Where possible, protocols should resist this attack.

Credentials versus implementations In typical public key-based systems, a peer's public key is authenticated using certificates (in the case of the protocols being discussed here, PKIX [13] certificates). Certificates are a general credential and are not tied to any specific revision of a given security protocol. Peers need to be able to communicate with agents that have a variety of combinations of new and old credentials and protocol capabilities.

In the remainder of this paper, we consider the application of these principles to two major Internet security protocols: S/MIME (a store-and-forward protocol) and TLS (a session-oriented protocol). A longer version of this paper, with analysis of other protocols and more details, appears in [1].

4 S/MIME

The first protocol we will consider is S/MIME [25, 26, 12]. S/MIME is a standard message encryption and authentication protocol. In the most common modes, it uses public key cryptography (RSA [14] and DH [28]) for key establishment, symmetric cryptography for bulk encryption,

	Receiver				
Sender	Old	Switch/Old	Switch/Both	Switch/New	New
Old	Old	Old	Old	-	
S/O	Old	Old	Old	Old	-
S/B	Old	Either	Either	Send New	New
S/N	-	New	New	New	New
New	-	New	New	New	New

Figure 1. Interoperability table for S/MIME implementations

and digital signatures (RSA and DSA [22]) for message authentication/nonrepudiation. User public keys are transported/authenticated using PKIX [13] certificates.

There are five major types of S/MIME client; the types of messages that each type of implementation should send are shown in Figure 1.

1. Old clients.
2. Switch-hitting clients with only old certificates.
3. Switch-hitting clients with both types of certificate.
4. Switch-hitting clients with new certificates.
5. New clients with only new certificates.

Note that this table assumes perfect information about the recipient's capabilities, which is not always the case. We now consider how to achieve interoperability in practice, which is a matter of trying to estimate the recipient's capabilities and create a message which they are most likely to be able to decode. For the remainder of this section, we focus on the behavior of Switch-hitting clients, since Old and New clients only have one possible behavior.

4.1 The Initial Message

The first case we consider is the case where a user is sending a message to someone with whom he has never communicated before. There are two possible sub-cases:

1. The sender does not have the recipient's certificate.
2. The sender has the recipient's certificate.

We consider each sub-case in turn.

4.1.1 Sending Without a Recipient Certificate

If the sender does not have access to the recipient's certificate, then he is subject to two limitations. First, he cannot encrypt because he does not have the public key to encrypt under. Second, he has no information about the recipient's capabilities, In particular, he cannot safely assume that the recipient's software will be able to process new hash functions.

Choice of certificate A sender with only one certificate must use that certificate. The difficulty comes when a sender has two certificates, one generated with an old hash function, and one with a new hash function. The possibilities, of course, are to use only one certificate or—because S/MIME allows multiple signatures—to use both. Any one-certificate strategy guarantees that some class of recipients will not be able to verify the message. Using both certificates preserves the possibility that the recipient can verify the message.

In order for this to work, however, recipients must be able to correctly verify messages with multiple signatures when one of them is unverifiable. Unfortunately, the S/MIME specification is fairly vague on this point. An unscientific poll of S/MIME implementors indicates that support for this option is spotty at best. [27, 11, 9].

Because receiver behavior is unpredictable, senders must attempt to estimate what sorts of implementation receivers are likely to have. This probably means choosing interoperability with the most popular strategies as a default (which are currently the older, weak, algorithms) and allowing users the option to configure a new behavior. This is irritating in that it involves a manual step if the sender guesses wrong. However there are already a number of non-security scenarios in which users must retransmit unreadable messages (bad attachment formats, HTML-vs-ASCII text, etc.) so it's not totally foreign to users.

Choice of digest algorithm Once the certificate has been chosen, the sender must choose a digest algorithm to digest the message before signing. This choice is made independently for each signature, so it is possible the message will be digested twice. In general, if the certificate being used was generated with one of the old algorithms (MD5, SHA-1), the message should be digested using SHA-1, which receivers are required to accept by section 2.1 of RFC 3851 [26]. This minimizes the chance that the recipient will not be able to verify the message signature. (MD5 should not be used at all for message digests, even if the certificate uses it.)

If the certificate being used was digested with a new hash algorithm, we recommend that the sender use the same algorithm to digest the message, on the grounds that if the recipient can use the digest algorithm to verify the certificate they can use it to verify the message. This runs the risk that the recipient will be using a separate toolkit to verify the certificate signature than they used to verify the message signature; however we are not aware of any S/MIME client that behaves in this way. This algorithm has the attractive property that it automatically works correctly with DSA, which can only sign SHA-1 digests.

4.1.2 Sending With the Recipient's Certificate

If the sender has the recipient's certificate(s) then the situation is simpler. We believe that it is a reasonable assumption that implementations can verify their own certificates and therefore must implement whatever digest algorithm was used to create them. If the recipient has only one certificate, the sender should therefore use their certificate with the corresponding algorithm. If the recipient has multiple certificates, the sender should use the one created using the strongest algorithm. For the reasons indicated above, we do not recommend sending multiple certificates in this case.

The choice of which certificate to send would be simpler yet if the recipient's certificate indicated which algorithms it was capable of using. Although this not currently possible the S/MIME working group is currently considering a draft [34] that would allow certificates to contain an SMIMECapabilities [26] extension for the owner of the certificate. This information could include information about allowed digest algorithms.

4.2 Subsequent Messages

Once an S/MIME implementation has received a signed message from a peer, it is in a much better position to estimate the sender's capabilities. For clarity, say that Alice has received a signed message from Bob. With high probability Bob can verify signatures produced with whatever algorithm(s) it used to digest its own message. If this is a new (strong) algorithm then all is good and Alice should herself use that algorithm.

If Bob used an old (weak) algorithm, then Alice at least knows that she can communicate with Bob using that algorithm. However, it is still possible that Bob has a Switch-hitting implementation. S/MIME has a standard way for Bob to signal this fact using the SMIMECapabilities signature attribute, which includes a (potentially partial) list of the algorithms that Bob supports. Bob can send a message using SHA-1 but include an SMIMECapabilities attribute indicating that he also supports SHA-512. If this attribute is included, it is always signed, thus preventing the introduction of a false attribute.

We recommend that when Switch-hitting implementations send messages using weak algorithms they include an indication that they also support a stronger algorithm. There is no point in including such an indication if you are sending with the stronger algorithm, since that algorithm is preferred and a recipient which cannot process the stronger algorithm cannot verify that you also support the weak one.

Because the SMIMECapabilities attribute is part of the signerInfo element, it is not included in messages which are unsigned. However, if Alice receives an encrypted message from Bob, she knows that he was able to verify the certificate that he used to encrypt to her. Therefore, if she wishes to sign future messages she should digest using whatever algorithm was used to produce that certificate.

4.3 Attacks

In this section, we consider the problem of protecting Switch-hitting implementations during the transition period when it is impractical to turn off support for the old algorithms. There are three basic scenarios:

- The attacker does not have a valid certificate and private key for either peer.
- The attacker has acquired a valid (but false) certificate and knows the private key.
- The attacker is one of the communicating parties.

4.3.1 Attacks Without a Valid Certificate

If the attacker does not have the private key for a valid certificate, then his ability to mount attacks, even on older digest algorithms, is fairly minimal unless he can compute preimages.[2] Clearly, an attacker who can compute preimages can undetectably modify messages in transit. In this case, the only defense is to stop using the affected algorithm. Note that senders cannot prevent this attack by multiply signing their messages; S/MIME multiple signatures are parallel and independent, so the attacker can simply strip the strong signature. Receivers must stop accepting an algorithm where computing preimages is possible.

4.3.2 Attacks Using a Valid Certificate

If the attacker has the private key for a certificate with a valid signature containing the identity of one of the peers—for instance obtained using an improved version of the Lenstra construction—he can impersonate that peer. This would allow him to forge messages that appear to be from that peer. It may also allow him to convince the other peer to encrypt messages using his fake certificates. The only certain countermeasure here is to stop accepting the compromised algorithm. One partial workaround would be for the victim to refuse to accept certificates dated after the time when the algorithm was compromised. This is a defense against collision attacks, but if the attacker can generate 2nd preimages, then he can forge a certificate with an arbitrary date and bypass this countermeasure. Another partial workaround is to store copies of previously used peer certificates (as with SSH [38, 39]), thus reducing the window of exposure to the first exchange of messages.[3]

[2] An attacker who can compute preimages is likely to be able to forge certificates. However, it is possible that an attacker could compute preimages but without fine enough control to forge a specific certificate.

[3] Note that it's common to store a digest of the certificate rather than the certificate itself. This obviously leaves one open to preimage attacks if the

4.3.3 The Attacker is One of the Communicating Parties

If it is easy to find collisions in a hash, then being one of the communicating parties—or at least in a position to substantially control the message contents—confers substantial advantage to the attacker. In particular, it allows him to cheat in contexts where an S/MIME signature is to be verified by a third party. The basic scenario is described in Section 2: two versions of a document are prepared, one innocuous and one malicious. One or both of the parties signs the innocuous version and then the attacker convinces the third party that the victim signed the malicious version. This attack can be mounted regardless of which party does the actual signing. The key is for the attacker to be allowed to prepare the document to be signed, since the colliding pair must be generated together.

In order to mount this attack on a Switch-hitting peer, the attacker must represent that he only supports the broken algorithm, thus forcing the signature to be performed using that algorithm. However, since supporting only old algorithms is a legitimate configuration, this is extremely easy to achieve. The victim has the choice of using that algorithm or not communicating at all.

This attack is extremely difficult to defend against in standard systems. Bob can defend against being conned by preparing the final document version and inserting enough randomness near the beginning (e.g., in a dummy field) to make it infeasible for Alice to have generated a collision.[4] However, this is complex and not supported by typical application software. Moreover, Alice should be suspicious of this request, since it allows Bob to mount a collision attack himself. A more general defense is for the parties to jointly agree on random values once the document content is fixed, but this is even more complex for ordinary users.[5] S/MIME implementations could of course do this automatically, but if one is willing to modify implementations it is easier to simply add strong algorithms.

We stress that this attack is very real and very practical if MD5 is used.

Because defense against this attack is difficult, in contexts when users are signing messages that might be verified by a third party, it is better to simply insist on using a strong algorithm. Similarly, third parties should be extremely suspicious when they are asked to rely on signatures that use weak algorithms, especially MD5. Note that as with the Lucks/Daum attack, close inspection of such messages generally will reveal their unusual structure and so this attack can only be mounted when the documents in question will be subject to only casual (or automatic) scrutiny.

5 TLS

TLS [6] is a standard channel security protocol which lives above the transport layer (where the OSI session layer sits). Originally designed for Web security [29], it is now widely used for other application protocols including SIP [33] and SMTP [17]. The most common TLS deployment involves an anonymous client connecting to a server and using the server's certificate and public RSA key for key exchange. There are five major places digest algorithms are used in TLS:

- In the per-record MAC.
- In the certificates used by client and server.
- In the digitally-signed element.
- In the PRF (pseudo-random function) used to make keying material.
- In the Finished message

TLS contains an extensive framework for algorithm negotiation, using the concept of "cipher suites". A cipher suite consists of a triple specifying the key establishment mechanism, the symmetric encryption algorithm used to encrypt traffic, and the message digest used to provide traffic message integrity. For instance, the cipher suite TLS_RSA_WITH_RC4_128_MD5 indicates RSA key exchange, encryption with RC4-128, and message integrity with a MAC based on MD5 (in TLS this is HMAC-MD5 [20].)

Unfortunately, this mechanism is only useful for negotiating the record MAC. Although there is a mechanism for negotiating client certificate type, it does not include digest algorithm and the other algorithms cannot be negotiated. Indeed, the PRF, ServerKeyExchange, and ClientVerify messages are not parametrized, but rather are specified directly in the standard. In order to accomodate newer digest algorithms in these cases we must extend TLS.

5.1 MAC Functions

Negotiating the MAC in TLS is straightforward. Each cipher suite specifies the digest function function to be used as the basis for the MAC. So, in principle all that needs to be done is to define a new set of cipher suites with stronger hash algorithms. Note that because TLS uses HMAC, the current collision-only attacks most likely do not represent a threat, thus making this a low priority upgrade.

attacker can manage to get a certificate with the same digest (not easy, because he must also simultaneously attack the CA's digesting process which covers different data). If a digest is being stored, it might be wise to store a keyed hash using some locally known key instead.)

[4]From a security perspective this is inferior to randomized hashing [10] but doesn't require changing the S/MIME implementation on either side.

[5]Kelsey and Kohno presented a "Herding" attack at the CRYPTO '05 rump session that allows cheating in this scenario, but the effort level (2^{87} for MD5, 2^{108} for SHA-1) far exceeds that of ordinary collision finding.

5.2 Server Certificates

The most important element of TLS to upgrade is the server certificate. Because certificates are automatically verified, they are the cryptographic technique most threatened by current digest attacks. TLS client certificates are rare; by contrast, virtually every TLS server has a certificate.

We assume that during the transition period, each server will have two certificates, one created with an old hash (typically SHA-1 or MD5) and one created with a new hash. The client can then indicate to the server that it can process the new certificate. There are two potential techniques for doing this: an overloaded cipher suite and a TLS extension [3]. The TLS extension approach is probably superior in that it preserves protocol cleanliness—the hash functions in the TLS cipher suite offers do not refer to the certificate. Moreover, there are performance reasons for the client to prefer to use the older hash algorithms for MAC functions: SHA-1 is much faster than SHA-256, and the MAC functions do not need to be upgraded immediately.

Note that this does not address the problem of DSA, which, as noted previously, cannot be used with any algorithm other than SHA-1. The cleanest solution for DSA is simply to to treat it as a new algorithm and define a new set of cipher suites that specify a newer version of DSA (e.g., DSA2).

5.3 Client Certificates

TLS client certificates are much less commonly used, although some organizations are using them. For example, the US government is now issuing client certificates for establishing user identities [18, 24]. However, in the case where client authentication is used, it is desirable to have a way for the server to indicate which hashes it would like the client to use. This is a fairly simply protocol engineering matter with two obvious alternatives:

- Add new values to the certificate_types field of the CertificateRequest message. For instance, an rsa_sign_sha256 type could be created.
- Use extension values.

Each of these approaches has advantages and disadvantages. The CertificateRequest approach keeps all the information about the certificates that the client should produce together but creates the risk of risk of combinatoric explosion of certificate_types values (only 256 such values are available). The alternative approach is for the server to use an extension indicating which hash algorithms it accepts. This is less elegant, but removes the combinatoric explosion problem. Neither approach is superior from a security perspective.

5.4 The Digitally-Signed Element

There are two places in TLS where data is explicitly digitally signed: the CertificateVerify and the ServerKeyExchange. In both places, the signature is accomplished using the "digitally-signed element". ("Digitally signed element" is the TLS term for a data element protected by a signature.) When the signature algorithm is DSA, the input is as expected—a SHA-1 digest of the data to be signed. However, when the signature algorithm is RSA, the input is something unusual: the MD5 and SHA-1 digests of the input are concatenated and fed directly into the RSA signature algorithm with PKCS#1 padding, but without DigestInfo wrapping. This is not a negotiatiable algorithm but rather is wired into the specification.

This unusual construction raises the question of what the target construction should be. The original rationale for the dual hash construction was to provide security in the face of compromise of either hash. However, in practice this has been partially undercut by the common heritage of SHA-1 and MD5. A practical attack on SHA-1 could potentially extend to compromising the MD5/SHA-1 pair. The general feeling in the TLS community is that a single negotiated digest would be a better choice.

The best choice here is probably to have the digitally-signed element use the same algorithm as was used to sign the certificate of the party doing the signing (the client for the CertificateVerify and the server for the ServerKeyExchange). This avoids the creation of a new negotiable option, thus reducing protocol complexity. In principle this could lead to interoperability problems if the certificate system has different capabilities than the TLS implementation. However, we're skeptical that the number of real implementations with this problem would be large enough to justify the additional complexity.

This change can either be implemented by having cipher suites that use strong algorithms (i.e., new cipher suites) use the newer digitally-signed construction or by changing the behavior of all cipher suites in a new version of TLS. Due to the low urgency of this change, we recommend the cleaner approach of creating a new TLS version.

5.5 PRFs

TLS uses a hash function-based PRF to create the keying material from the PreMaster Secret and Master Secret. It is also used to compute the Finished messages which are used to secure the TLS negotiation against downgrade attack. Compromise of the PRF might potentially allow an attacker to determine the keying material or mount a downgrade attack.

The TLS PRF is actually two PRFs, both based on HMAC, with one using MD5 and the other using SHA-1.

Like the digitally-signed element, the TLS PRF is explicitly specified in the standard and not negotiable.[6] This construction, while somewhat over-complex, is provably secure under the assumption that either HMAC-SHA1 or HMAC-MD5 are secure pseudorandom functions [19]. Because the current attacks do not affect the security of HMAC, upgrading the PRF is a low-priority task. However, we briefly consider methods here.

The two basic methods for negotiating the PRF algorithm are to use the negotiated cipher suite or to create a new extension. In the first case, whatever digest algorithm was negotiated for the cipher suite would also be used as the basis for the PRF. This has the obvious drawback that it ties TLS to the basic HMAC-X structure of the PRF. If this construction were found to be insecure (despite the proofs of security), then it would not be possible to negotiate a new construction. By contrast, while using an extension adds complexity it would allow substitution of the construction without creating a new version of TLS.

We are skeptical that this increased flexibility justifies the added complexity of defining a new extension. In view of the security proofs for HMAC and its wide use in TLS, it seems likely that any attack on HMAC would imply compromise of the underlying digest function and result in the compromise of key elements of the system (message MACs, certificates, etc.), thus necessitating a new revision of TLS in any case. It would be straightforward to revise the PRF at that time.

PRFs have similar roll-out issues to those described in Section 5.4. As with the digitally-signed element, we recommend that the transition to a negotiated PRF occur in a future version of TLS.

5.6 The Finished Message

The TLS Finished message is computed by computing the TLS PRF over the master secret and the concatenation of two digests over the handshake messages, one using MD5 and one using SHA-1. The same considerations apply here as in the PRF. The hash itself is unkeyed although both sides contribute random nonces. This design modestly reduces memory requirements on the client and server; HMAC-based MACs require having the key available at the beginning of the MAC computation, but the key is only available after the key exchange, so using HMAC directly would require storing initial handshake messages. The hash-then-PRF technique only requires storing the hash state. There is a potential risk in this design in that keyed hashes are harder to attack than simple hashes. However, because the attacker cannot control the client messages and can only slightly influence the server's messages (by modifying the client messages in flight to produce a different negotiation result) the ability to create collisions is insufficient to mount this attack.

The obvious approach to transition is to replace the pair of hashes with the negotiated hash function used for the message MAC. However, note that this requires both sides to store the handshake messages until the MAC algorithm is decided (in the ServerHello). This requires a modest change in TLS implementation behavior and a slight increase in storage requirements. An alternative design would be to replace the "digest then PRF" construction with a MAC directly over the handshake messages. This would have only slightly higher storage requirements and be modestly more secure in the event of preimage attacks on the underlying hash function. We consider either approach adequate, though we believe that the security considerations outweigh the memory issue and therefore recommend transitioning to a simple MAC over the messages.

5.7 Attacks

As with S/MIME, we consider the problem of protecting Switch-hitting implementations during the transition period. The general form of the attack is for the enemy to force one or both sides to believe that the other side is an old implementation and convince them to use weaker algorithms, thus rendering them susceptible to attack.

We can divide these attacks broadly into two categories. In the first, the attacker has obtained a valid certificate for one side of the connection (most likely the server) and knows the corresponding private key. In this case, no complete defense is possible other than turning off the old algorithm. The attacker can simply intercept the connection and use its certificate. As with S/MIME, partial defenses including rejecting newer certificates signed with weak algorithms and SSH-style fingerprint comparison.

If the attacker does not have a valid certificate, he must attack the negotiation more indirectly. However, because the negotiation is protected by a MAC computed using the PRF, the attacker must be able to predict PRF output in order to predict the key used for the PRF. As argued in Section 5.5, this would require a very serious break of HMAC and most likely that the attacker can compute preimages, making a direct attack on certificates possible.

6 Design Principles for Algorithm Agility

It is clear from our analyses that designing for algorithm agility is harder than thought. In this section, we present some suggestions for protocol design that may make future transitions smoother.

[6]This has already been an issue with the proposed GOST cipher suite [5], which for regulatory reasons must use the GOST digest function in the PRF

6.1 Avoid the use of hardwired cryptographic algorithms

Any protocol which depends on a single hardwired algorithm is inherently brittle—if that algorithm is broken it can be very hard to repair the protocol. This is particularly obvious in the case of DSA; the transition from SHA-1 to some other digest algorithm is going to be much more difficult than with RSA, because the hash algorithm and the signature algorithm will need to be replaced simultaneously. Similarly, the decision to hardwire MD5 and SHA-1 into the basic structure of SSL/TLS necessitates far more protocol re-engineering than if the algorithms had been parametrized in the first place.

6.2 Provide mechanisms for capability discovery and/or negotiation

Even if a protocol allows for the use of multiple algorithms, algorithm transitions can be difficult to accomplish if the agents do not have good information about the capabilities of the peers with which they wish to communicate. In session-oriented protocols such as TLS or IPsec, this information is easy to exchange by incorporating an algorithm negotiation phase in the session establishment. In store-and-forward protocols, however, the problem is much more difficult and generally requires some sort of directory which can be used by agents to advertise their capabilities. In both kinds of protocols, it is important to allow for the advertisement of capabilities for every parameter. S/MIME, TLS, and IPsec all fail this test.

6.3 Capability discovery should occur as early as possible

Even protocols which allow negotiation often fail to do so early enough in the exchange. In the best case, one party must try to adapt to whatever is eventually chosen, possibly resulting in increased computational costs (as with TLS CertificateVerify messages or S/MIME multiple signatures). In the worst case, one side must guess about the other side's capabilities and a wrong guess results in non-interoperability or a requirement for manual reconfiguration (as with S/MIME single signatures or IPsec hash functions). In order to avoid this, protocol designers should allow negotiation/capability discovery to happen as early in the communication process as possible.

There is a tradeoff to consider here. If discovery is done entirely before crypto, it can introduce extra latency (e.g., one round trip for the discovery/negotiation, then another for the crypto). Careful protocol design can mitigate this to some extent: for instance, the hash negotiation technique described in Section 5.4 implies some cryptography before algorithm negotiation but can be implemented with only a small working buffer. IKE already includes a capability discovery exchange in the right place; the problem is that some necessary capabilities were not negotiated. Another approach is to design protocols where one peer is optimistic but can fall back if they guess wrong, as in [35].

6.4 Avoid downgrade attacks

Downgrade attack is a persistent problem with negotiation in security protocols. There are generic techniques for protecting negotiation in session-oriented protocols (typically by computing a MAC over the handshake messages) but they cannot be guaranteed to work if the MAC or key exchange algorithms are broken. Although TLS and IPsec do a fairly good job of this via HMAC, which is largely unaffected by the current level of attacks, they are still at risk if hash compromises lead to compromise of the certificate system used to authenticate the key establishment phase. The downgrade problem is substantially harder with store-and-forward protocols. For instance, in the case of S/MIME multiple signatures the attacker can simply delete the stronger signature. In general, although defenses against downgrade are important to incorporate, sufficiently powerful attacks on the cryptographic algorithms will require agents to stop accepting those algorithms.

7 Conclusions

It is clear that new hash functions or new methods of employing hash functions are necessary. However, as we have demonstrated, neither the specifications nor implementations are ready for the transition. We have presented an analysis of transition strategies for S/MIME and TLS; analysis of other protocols, including IPsec and DNSsec, appears in [1]. We strongly urge the analysis of other protocols that use hash functions. Prominent candidates include OpenPGP [4], and Secure Shell [38, 39].

For the protocols we analyzed, we present recommendations to implementors and the IETF. These changes are necessary *to prepare* for the transition. We suggest that they be made as quickly as possible, to provide maximum secure interoperability when new hash functions are ready.

When protocol upgrades are being designed, consideration should be given to signature algorithm agility as well. In most cases, the signaling will have to be done in the same place as for hash functions. However, some of the overloading options are inappropriate for signature algorithms. For example, in IPsec one might use the appearance of a new hash algorithm in the SA proposal as a signal that one party supports a new hash algorithm in one context, and hence presumably in another. There is no obvious way to extend this to, say, support of ECC signatures.

S/MIME

Implementors of S/MIME should ensure that their product handles multiple signatures properly. In particular, programs should report success with one signature while warning about unverifiable signatures.

MD5 should *never* be used for digests, since all conforming implementations already support SHA-1.

The IETF should develop a method for indicating digest function capabilities in certificates, CA vendors should implement it, and new certificates should contain explicit statements about hash functions supported.

TLS

The IETF should define a TLS extension by which servers can signal support for newer certificates.

The IETF should pick one of the two suggested alternatives for supporting client side certificates properly.

The IETF should consider making the PRF depend on the MAC algorithm in a future version of TLS.

The definition of the digitally-signed element should be amended to support new hash functions.

The definition of the Finished message should be amended to support new hash functions.

DSA

DSA presents a special problem, since it may only be employed with SHA-1. NIST needs to clarify this situation, either by defining DSA-2 or by describing how DSA can be used with randomized hashes or truncated longer hashes.

The problems we have described here are symptomatic of a more general problem. Most security protocols allow for algorithm negotiation at some level. However, it is clear that this has never been thoroughly tested. Virtually all of the protocols we have examined have some wired-in assumptions about a common base of hash functions. It is a truism in programming that unexercised code paths are likely to be buggy. The same is true in cryptographic protocol design.

Acknowledgments

The authors would like to thank Russ Housley for his detailed review of this paper. We would also like to thank Alex Alten, Steve Burnett, Hal Finney, Eu-Jin Goh, Peter Gutmann, Steve Henson, Charlie Kaufman, John Kelsey, Ben Laurie, Arjen Lenstra, Blake Ramsdell, Hovav Shacham, Benne de Weger, and the referees for their advice and comments.

References

[1] S. M. Bellovin and E. K. Rescorla. Deploying a new hash algorithm. Technical Report CUCS-036-05, Dept. of Computer Science, Columbia University, October 2005.

[2] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and Reduced SHA-1. In *Proceedings of Eurocrypt '05*, 2005.

[3] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright. Transport Layer Security (TLS) Extensions. RFC 3546, June 2003.

[4] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP Message Format. RFC 2440, November 1998.

[5] G. Chudov and S. Leontiev. Addition of GOST ciphersuites to Transport Layer Security (TLS), May 2004. draft-chudov-cryptopro-cptls-01.txt.

[6] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, January 1999.

[7] H. Dobbertin. Cryptanalysis of MD4 (Third Workshop on Cryptographic Algoruthms, Cambridge 1996). *Lecture Notes in Computer Science*, pages 55–72, 1996.

[8] H. Dobbertin. The First Two Rounds of MD4 are Not One-Way. *Lecture Notes in Computer Science*, 1372, 1998.

[9] P. Gutmann. Personal communication, 2005.

[10] S. Halevi and H. Krawczyk. Strengthening Digital Signatures via Randomized Hashing, May 2005. draft-irtf-cfrg-rhash-00.txt.

[11] S. Henson. Personal communication, 2005.

[12] R. Housley. Cryptographic Message Syntax (CMS). RFC 3852, July 2004.

[13] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280, April 2002.

[14] J. Jonsson and B. S. Kaliski. Public-key cryptography standards (PKCS) #1: RSA cryptography specifications version 2.1. RFC 3447, Internet Engineering Task Force, Feb. 2003.

[15] A. Jøsang, D. Povey, and A. Ho. What you see is not always what you sign.

[16] K. Kain, S. Smith, and R. Asokan. *Digital Signatures and Electronic Documents: A Cautionary Tale*. Kluwer Academic Publishers, 2002.

[17] J. Klensin. Simple Mail Transfer Protocol. RFC 2821, April 2001.

[18] L. D. Kozaryn. DoD issues time-saving Common Access Cards, 10 October 2000. American Forces Information Service, `http://www.defenselink.mil/news/Oct2000/n10102000_200010107.html`.

[19] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticaticated Diffie-Hellman and its Use in the IKE Protocol, June 2003. `http://www.ee.technion.ac.il/~hugo/sigma.ps`.

[20] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997.

[21] A. Lenstra, X. Wang, and B. de Weger. Colliding X.509 Certificates. In *Proceedings of ACISP*, 2005. To appear. Online: `http://eprint.iacr.org/2005/067`.

[22] National Institute of Standards and Technology, U.S. Department of Commerce. Digital Signature Standard, 2000. FIPS PUB 186-2.

[23] National Institute of Standards and Technology, U.S. Department of Commerce. Secure Hash Standard, 2002. FIPS PUB 180-2.

[24] National Institute of Standards and Technology, U.S. Department of Commerce. Personal identity verification (PIV) of federal employees and contractors, 2005. FIPS PUB 201.

[25] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling. RFC 3850, July 2004.

[26] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851, July 2004.

[27] B. Ramsdell. Personal communication, 2005.

[28] E. Rescorla. Diffie-Hellman Key Agreement Method. RFC 2631, June 1999.

[29] E. Rescorla. HTTP Over TLS. RFC 2818, May 2000.

[30] E. Rescorla. MD5 Collisions in PostScript Files", June 2005. `http://www.educatedguesswork.org/movabletype/archives/2005/06/md5_collisions.html`.

[31] R. Rivest. MD4 Message Digest Algorithm. RFC 1186, October 1990.

[32] R. Rivest. The MD5 Message-Digest Algorithm . RFC 1321, April 1992.

[33] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, June 2002.

[34] S. Santesson. X.509 Certificate Extension for S/MIME Capabilities, May 2005. draft-ietf-smime-certcapa-05.txt.

[35] H. Shacham, D. Boneh, and E. Rescorla. Client-side caching for TLS. *ACM Transactions on Information and System Security (TISSEC)*, 7:553–575, Nov 2004.

[36] X. Wang, Y. Yin, and H. Yu. Collision Search Attacks on SHA1, 2005. `http://theory.csail.mit.edu/~yiqun/shanote.pdf`.

[37] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions. In *Proceedings of Eurocrypt '05*, 2005.

[38] T. Ylonen. SSH – secure login connections over the Internet. In *Proceedings of the Sixth Usenix Unix Security Symposium*, pages 37–42, July 1996.

[39] T. Ylonen. SSH protocol architecture, 2005. draft-ietf-secsh-architecture-22.txt.