SKYPE SECURITY EVALUATION

Tom Berson
Anagram Laboratories
18 October 2005

berson@anagram.com
skype:tomnd2t

1. Introduction and Summary

I have been a Skype user since August 2004. My 35-year long career as cryptographer and computer security expert has taught me to be professionally skeptical about the security of almost everything, especially of a system which is as adept as Skype at getting through typical network defenses. So I re-formatted the hard disk on a spare computer and dedicated the box to the Skype application. Over the next few months I monitored the list of processes running on the machine, looking for anything suspicious. I also ran a number of experiments during which I captured and analyzed the packets flowing into and out of the box. I was looking for malicious activity and trying to figure out how Skype works. Perhaps you have run similar experiments yourself.

You may imagine my delight when, in April 2005, Skype contacted me and invited me to compete for the job of performing an independent evaluation of Skype information security, with a special focus on the Skype cryptosystem. I traveled to the Skype engineering center in Tallinn, and to the Skype business center in London. In each place I interviewed Skype people and was interviewed by them. The meetings went well; I won the business. Since 1 June 2005 I have been analyzing the security properties of Skype software and services, with a focus on the current and planned uses of cryptography. I have had unimpeded access to Skype engineers and to Skype source code. I have found out a lot about Skype. The more I found out, the happier I became.

Skype makes wide use of cryptography to authenticate user and server identities, and to protect the content transmitted across the P2P network from disclosure by parties other than the peers. The cryptographic systems engineered for these purposes are well-designed and correctly implemented. The goals of providing verified user identity and content confidentiality across the P2P cloud are achieved. I believe Skype can be proud of its intelligent and correct use of cryptography toward these ends.

Skype uses only standard cryptographic primitives to meet its ends, which is a sound engineering approach. These primitives include the AES block cipher, the RSA public-key cryptosystem, the ISO 9796-2 signature padding scheme, the SHA-1 hash function,

and the RC4 stream cipher. I looked at the Skype implementation of each of these, and verified that each implementation conforms to its standard and interoperates with reference implementations.

Skype operates a certificate authority for user names and authorizations. Digital signatures created by this authority are the basis for identity in Skype. Skype nodes entering into a session correctly verify the identity of their peer. It is infeasible for an attacker to spoof a Skype identity at or below the session layer. (I have not examined any higher layer code).

Skype uses a proprietary session-establishment protocol. The cryptographic purposes of this protocol are to protect against replay, to verify peer identity, and to allow the communicating peers to agree on a secret session key. The communicating peers then use their session key to achieve confidential communication during the lifetime of the session. I analyzed this protocol, and found that it achieves its cryptographic aims. Further, I explored the strength of the protocol against a range of well-known attacks, including replay attack and man-in-the-middle attack. I determined that each of the attack scenarios is computationally infeasible.

1.1 Caveats

This report represents a four-month evaluation. A longer evaluation effort might uncover problems not yet seen. The Version 1.3 code base was evaluated. The code base continues to evolve beyond that snapshot.

1.2 Security Policy

A Security Policy defines what "security" means in the context of a system and allows one to answer the question, "Is this system secure?" A security policy is a great help to designers, implementers, operators, managers, and users of a system. The Skype Security Policy is:

1. Skype usernames are unique.

2. Users or applications must present a Skype username and its associated authentication credential (*e.g.*, password) before they exercise that username's identity or privileges.

3. Each peer correctly provides the other with proof of its username and privileges whenever a Skype session is established. Each verifies the other's proof before the session is allowed carry messages (*e.g.*, voice, video, files, or text).

4. Messages transmitted through a Skype session are encrypted from Skype-end to Skype-end. No intermediary node, if any exist, has access to the meaning of these messages.

2. Overview of Skype Cryptography

2.1 Registration

The central cryptographic secret in Skype is the Central Server's private signing key, S_S. The corresponding public verification key, V_S, and an identifier for the key pair are installed in every Skype client at build time.

Enrolment in the Skype cryptosystem begins with user registration. The user selects a desired username, call it A, and a password, call it P_A. The user's client generates an RSA key pair, (S_A and V_A). The private signing key, S_A, and a hash of the password, $H(P_A)$, are stored as securely as possible on the user platform. (On the Windows platform this is done using the Windows CryptProtectData API).

The client next establishes a 256-bit AES-encrypted session with the Central Server. The key for this session is selected by the client with the aid of its platform-specific random number generator. The client can and does verify that it is really talking to the server. The client sends the server, among other things, A, $H(P_A)$ and V_A.

The Central Server decides whether A is unique, and otherwise acceptable under Skype naming rules. If so, the server stores (A, $H(H(P_A))$) in a database. It forms and signs an Identity Certificate for A, IC_A, which contains, among other things, the Central Server's RSA signature binding A and V_A, $\{A, V_A\}^S_S$ and the key identifier of S_S. IC_A is returned to A.

Actually, the above discussion of the server signing key is a simplification for purposes of clarity. In fact, there are two Central Server key pairs, one with a modulus of 1536 bits and the other with a modulus of 2048 bits. The choice of which modulus to use is made by the Central Server. It depends on whether the enrolling user has purchased any Skype premium services, *e.g.* SkypeOut. If so, the longer modulus is used. If not, the shorter modulus is used. An enrolled user who purchases a premium service for the first time will be issued a new IC, signed with the longer key.

There is another simplification going on in the above discussion. The Central Server in fact consists of a number of machines with different functions, including one machine which does nothing but sign certificates. Also, the entire Central Server pod is replicated several times over for performance and business continuity.

2.2 Peer-to-Peer Key Agreement

Suppose now a caller, A, wishes to communicate with callee, B, and there is no pre-existing Skype session between them. In this case a new session is established and provided with its own 256-bit session key, SK_{AB}. This session will exist so long as there is traffic in either direction between A and B, and for some fixed time afterward. After the session is over, the SK is retained in memory until the client is closed, at which time it is zeroed.

Session establishment first requires establishing connectivity between A and B across the Skype cloud. Using this connectivity, A and B now engage in a key-agreement protocol during which, among other things, they check for freshness, verify each other's identity, and agree on SK_{AB}.

2.3 Session Cryptography

All traffic in a session is encrypted by XORing the plaintext with key stream generated by 256-bit AES (also known as Rijndael) running in integer counter mode (ICM). The key used is SK_{AB}. Skype sessions contain multiple streams. The ICM counter depends on the stream, on salt, and the sequency within the stream.

3. Details of Skype Cryptography

3.1 Random Number Generation

Random numbers are used for several cryptographic purposes within Skype, such as protection against playback attacks, generation of RSA key pairs, and generation of AES key-halves for content encryption. The security of a Skype P2P session depends importantly on the quality of the random numbers generated by the communicating peers.

Random number generation varies from platform to platform. So far, I have evaluated random number generation only on the Windows platform, where it is done well. Platforms with more limited processing power or more limited internal state can be expected to be challenging environments for random number generation, and these might well be productively evaluated in the future.

On a Windows operating system platform, Skype makes Win32 system calls to a number of operating system functions. The bits gathered from these calls, together with some salt, are hashed using SHA-1. The high-order 64-bits of the hash result are returned.

This procedure is successful in gathering sufficient entropy to return 64 bits. The entropy gathering and mixing of the bits follows standard recommended practices as described in RFC 1750. The methods are also very similar to the Microsoft CryptoAPI function, CryptGenRandom, which is described in *Writing Secure Code*, 2nd Edition, Microsoft Press, pp. 262-269.

3.2 Cryptographic primitives

Skype uses standard cryptographic primitives to achieve its security goals. There is no proprietary encryption in Skype, which is a good engineering practice. Standard primitives have the advantage of past and ongoing analysis and evaluation throughout the world. The cryptographic primitives used in Skype are: the AES block cipher, the RSA public-key cryptosystem, the ISO 9796-2 signature padding scheme, the SHA-1 hash function, and the RC4 stream cipher.

3.2.1 AES

I evaluated the Skype AES (Advanced Encryption Standard) code in a stand-alone environment.

The code correctly implements AES using a block size of 128 bits and a key size of 256 bits. Standard AES-256 test vectors and keys were compared to the Skype AES. Skype AES matches the results produced by other implementations. Skype put significant effort into making Skype AES run quickly. It is optimized for Integer Counter Mode. It uses macros to speed performance. I compared Skype AES code to two other optimized C/C++ implementations. The Skype encrypt function performs favorably in terms of clock cycles per encryption.

Skype AES, in Integer Counter Mode (ICM), is used as a key generator for data packet encryption. A buffer with the plaintext (packet data) in all but the last two bytes is encrypted as follows:

A. Successive blocks of plaintext are XORed to AES cipher blocks. The latter are generated using a key established for the session. The input (counter) blocks are concatenations of

salt : salt : packet_index : block#

The packet_index is a 48-bit value and the block# is a 16 bit value.

B. A CRC is computed on the contents of the encrypted buffer. The mod 2 sum of the CRC with the low order 2 bytes of the packet_index is stored in the last 2 bytes of the buffer.

C. Note: Only the low order bits of the AES counter change from block to block while encrypting a buffer. The packet index changes from buffer to buffer. The salts are contributed by each peer and are random values.

3.2.2 RSA

The Skype code which tests for primality and generates key pairs appears to be implemented correctly. The code uses the odd powers variant of the standard square-and-multiply algorithm to perform modular exponentiation, and also uses smart squaring (which cuts the number of multiplication operations in half). In addition, the code implements the critical parts in assembly language where possible. This is platform-dependent.

The Miller-Rabin test in the prime number generation code includes all the necessary test conditions of Miller-Rabin. The default number of iterations (25) included in the test makes the chances of misidentifying a composite number as prime extremely low (probability < 10^{-16}). Even 5 Miller-Rabin iterations will yield a probability = 0.00063 of accepting a composite number. This is still an acceptable value and reduces the time to generate the primes on a client machine with limited processor speed

I observed an opportunity to improve the efficiency of primality testing, and communicated this to the Skype engineering team.

The algorithm to generate the decryption exponent (private key) is a correctly implemented Montgomery method variant of modular inversion. This method, although it uses extra computations, eliminates the expensive trial divisions required by the Euclidian method, and replaces expensive normal divisions with the much cheaper divisions by two.

Several of the multi-precision integer methods used for RSA calculations of primes and key pairs include assembly code for specific processors (ARM and x86) to improve efficiency.

I tested the key pair generation, encryption, and decryption processes and found that they generate correct results.

3.2.3 Signature padding

The RSA signature padding code is compliant with ISO 9796-2. For smaller payloads, the padding takes either of the below forms:

4A <data><sha1(data)> BC
4B BB…..BA <data><sha1(data)> BC

Larger payloads are split up and each portion is padded in the following format:

6A <partial data><sha1(complete-data)> BC

The signature verification method checks the integrity of the signed message. It decrypts the RSA and extracts and checks the padding. It also checks the hash for accuracy. Consistent with ISO 9796-2, after the first signed block, the rest of the signed message is in plaintext, and this is verified via the SHA-1 hash check.

3.2.4 SHA-1

The code implementing the Secure Hash Algorithm (SHA-1), is beautiful and tight. In fact, this version is easier to follow than the openssh source code SHA-1 implementation. There were not any observable type mismatches, buffer problems, etc. The code compiles cleanly with no warnings.

There are two interfaces to the hash function. Both were tested with no problems. The Skype SHA-1 code is correct. It passes its own test vectors, and other published test vectors. It passes Jim Gillogly's word roll-over tests. I wrote a script to check Skype SHA-1 output against Perl's SHA-1 implementation for some large randomly generated buffers. Nothing anomalous was identified during these tests.

3.2.5 RC4

The RC4 algorithm is used in Skype to generate putative primes for RSA. The implementation of RC4 is standard. The initialization of RC4 with random seed bits and use of the RC4 key stream for generating RSA keys is an acceptable technique. A similar usage is exhibited by Microsoft in their CryptGenRandom() function.

3.3 Peer-to-Peer Key Agreement Protocol

Key-agreement is achieved using a proprietary protocol. I constructed a formal model of the protocol, and analyzed the model for protocol flaws. I also verified that the protocol is correctly implemented in the source code.

The protocol is symmetric. Neither party is at an advantage; each is equi-potent and receives identical assurances.

To protect against playback, the peers challenge each other with random 64-bit nonces, and respond by returning the challenge, modified in a standard way, and signed with the responder's private signing key.

To establish identity, the peers exchange their Identity Certificates (signed by the Central Server) and verify that these certificates are valid. Because an Identity Certificate contains a public key, each peer can then verify signatures formed by the other party. Also, each peer can RSA-encrypt a message for the other party alone.

Each party contributes 128 random bits toward the 256-bit session key. The contributions are exchanged as RSA cryptograms. The two contributions are then combined in a cryptographically-sound way to form the shared session key.

3.4 Attacks on the Skype Key Agreement Protocol

One way to examine the strength of any key-agreement protocol is to explore the feasibility of various attacks against it. I considered attacks against single instances of the protocol, and also attacks against multiple simultaneous instances of the protocol.

3.4.1 Man-in-the-Middle (MITM) Attacks

The goal in this attack is for an intermediary attacker, the MITM, to impersonate the caller and/or the callee to each other. Then, information would be passed from the caller to the attacker to the callee and vice versa. The goal of this attack is access to the entire communication of the caller and the callee, as well as ignorance by the caller and the callee that the eavesdropping had occurred.

In order to perform a MITM attack, the attacker must be able to convince the caller that she is the callee (and vice-versa). The attacker can do this with a valid signed certificate that indicates her username is the callee (resp. caller). This certificate can either be the real certificate used by the callee (resp. caller), or a forged certificate. The attacker must also be able to intercept and/or block all traffic between the caller and the callee.

Assuming this ability, I explored several attack scenarios.

- One scenario prevents a session from being established, but does not compromise confidentiality of the communications.

- Two other scenarios require either the defeat of the physical, hardware, and software security mechanisms at a participating peer or an infeasible pre-computation. With that preparation, these two scenarios then require some intercept followed by a second infeasible post-computation. If all that could be done, the attacker could compromise the security of a single peer-to-peer session.

- Another scenario requires defeat of the security at both peers. In this case, all sessions between that pair of peers can be compromised.

- A last scenario requires defeat of the security mechanisms at the Skype Central Server. As I pointed out above, digital certificates created by the certificate authority are the basis for identity in Skype.

3.4.2 Replay Attack

A replay attack seeks to convince a node to enter into a session with an attacker by replaying data captured by the attacker from a previous session between the target and another node. Possible goals of a replay attack include duplicating a key stream used previously (which may enable cryptanalysis-at-depth), and blocking a node from communicating with a certain other client.

The attacker could observe multiple handshakes involving a target node. This would give access to multiple challenges and responses. The attacker could then send a challenge to the target pretending to be a previous peer. The target would respond with a challenge of its own. If the target challenge was identical to one that the attacker had observed previously for this caller, the attacker could then answer the challenge correctly and proceed to the next aspects of the key exchange protocol. However, because challenges are 64 bits long and chosen at random, the probability of this happening is low. The chance of getting a challenge repeat from the client is, in the case of few observations, the number of observations N over the total possibilities, $N / 2^{64}$.

Even if this unlikely event occurred, the attacker would still not have access to the AES key unless the even more unlikely event occurred that the target chose at random the same 128 bit key contribution as it chose during the session that the attacker recorded. This might happen once every 2^{128} tries, a vanishing low probability. And even this is an overstatement because it does not consider the salutary effects of salting the counter

3.4.3 Password Guessing Attack

Users get to choose whether to “remember” their Skype password on the platform they are using. Most users choose this option. On the Windows platform, the password is given to the operating system to protect under the Windows CryptProtectData API. A user who can later login to Windows can use Skype without further presenting any credentials. The minority of users who choose not to remember their password on the computer they are using must login via a client-server protocol before they can use Skype. To protect against password-guessing or dictionary attacks, the Skype Central Server forces a timeout after a series of incorrect passwords.

3.4.4 Weakness in use of CRC

CRC-type checksums are commonly used in communication protocols to reliably and efficiently detect bit errors. However, because they are linear, they may be unsuitable for

the purpose of detecting intentional modification of data. This was one problem discovered in WEP, the original security protocol for IEEE 802.11 wireless LANs. Some aspects of Skype use CRC type checksums in a manner similar to WEP and consequently with some of the same weaknesses.

This problem has been reported to Skype and is scheduled to be repaired in an upcoming release.

3.4.5 Side-Channel Attacks

It is well known that implementations of cryptographic operations may sometimes leak information about the plaintext or the key through their consumption of shared resources, such as storage, CPU time or power. The Skype client makes no defense against this sort of attack. So, for example, if a malicious program were running on the same platform as a Skype client, that malicious program might be able to deduce bits of the user's private signing key. This would eventually allow the owner of the malicious program to masquerade as the user. I consider this a minor problem, because a malicious program running on the same platform as a Skype client could do much greater damage directly.

3.4.6 ASN1 Attack

A few years ago, a group of Finnish researchers at Oulu University discovered a variety of potentially dangerous vulnerabilities in the SNMP agents of a number of prominent vendor's products. The most common source of these difficulties was in these products' inability to safely and correctly parse ASN1 encoded payloads. Not surprisingly, such problems carried over to SSL's use of X509 certificates, as well as that of other protocols relying on some manner of that encoding scheme.

Skype protocols do not use ASN1, but they do employ a similar mechanism and rely heavily on their ability to correctly parse encoded payloads. Included in these payloads are fields which an attacker may set to almost any value. So it is imperative that the Skype code which parses these payloads be correct. I reviewed the Skype payload parsing code. I found a potential error associated with the decoding of integers. The error does not imperil the confidentiality of Skype communications, but might lead to unpredictable behavior in the presence of malicious inputs. I communicated this information to Skype engineering.

4. The Bottom Line

I started as a skeptic. I thought the system would be easy to defeat. However, my confidence in the Skype grows daily. The more I find out about it, the more I like.

In 1998 I observed that cryptography was changing from expensive to cheap, from arcane to usual, from difficult to easy, from scarce to abundant. My colleagues and I tried to predict what difference, if any, abundant cryptography might make in the world. What new forms of engineering, business, economics, or society would be possible? We did not predict Skype. But, now that I am coming to know it well, I recognize that Skype is an early example of what abundant cryptography can yield.

The designers of Skype did not hesitate to employ cryptography widely and well in order to establish a foundation of trust, authenticity, and confidentiality for their peer-to-peer services. The implementers of Skype implemented the cryptographic functions correctly and efficiently. As a result, the confidentiality of a Skype session is far greater than that offered by a wired or wireless telephone call or by email and email attachments.

Beyond errors in the cryptosystem, I have also looked for back doors, Trojans, overreaching “debugging” facilities, etc. I did not find any hints of malware in the portions of the Skype code I reviewed.