Security Performance

Daniel A. Menascé • *George Mason University* • *menasce@cs.gmu.edu*

Threats can arise at many different points in modern networked systems and compromise one or more security dimensions, including integrity, confidentiality, authentication, nonrepudiation, availability, and privacy. To better understand the threats, we must examine what each of these dimensions means. Integrity, for example, indicates that data are protected from unauthorized modification. Confidentiality ensures that only authorized users can read data. Authentication ensures that the other party in a dialogue is not an impostor. Nonrepudiation prevents someone from denying having sent a message. Availability ensures that a system is immune to denial-of-service attacks, which can prevent access by legitimate users. Finally, privacy provides controls on how information about individuals will be used and disseminated.

Several protocols and mechanisms aims to enforce the various dimensions of security in applications ranging from email to e-commerce transactions. Most use encryption as a basic component. Adding such mechanisms and procedures to applications and systems does not come cheaply, however. As Ravi Sandhu points out, security is intrusive and increases the total cost of computer-system ownership.[1] He also indicates that security is about trade-offs, rather than absolutes, and that we should strive for good-enough security, not for more security than necessary. This thought underlies the following description of some security trade-offs in the areas of performance and scalability. My analysis is quantitative in nature and aims to illustrate the effect of using a specific security protocol or set of cryptographic algorithms on a computer system's performance.

Basic Security Operations

Security protocols use one or more basic security operations, such as symmetric key cryptography, public and private key cryptography, and secure hash functions. As Figure 1 shows, in symmetric encryption, both parties must use the same key to encrypt and decrypt a message. This poses a challenge in key management and distribution because the number of keys grows with the square of the number of communicating parties. Examples of symmetric key encryption algorithms include the data encryption standard (DES), 3DES, RC4, RC5, and the advanced encryption standard (AES).[2]

In public key encryption, each communicating party must have a pair of keys: a private one, which must be securely held, and a public one, which anyone can have. If encryption is performed with someone's public key, decryption must be done with the corresponding private key. The reverse is also true: if a message is encrypted with someone's private key, it must be decrypted with that person's public key.

Symmetric key encryption is orders of magnitude faster than public key encryption. For example, encrypting a 128-byte block using a public key of 512 bits takes 3.5 milliseconds on a Pentium-II 266 MHz[3] whereas symmetric key encryption using AES takes less than one microsecond on the same machine.[4]

Private key operations are generally slower than those with public keys. As Figure 2 shows, private key operations can be more than 38 times slower than public key operations with a 2,048-bit key. The ratio between private and public key operation times grows almost linearly with the key length k. However, the time to perform a public or private key operation increases with a power of the key length k. For the data used in Figure 2, for example, the time required to perform a private key operation on a 128-byte block increases with $k^{2.75}$; the time needed to perform a public key operation on the same size block increases with $k^{1.87}$. Longer keys provide an increased level of security but at a performance cost.

Another security-protocol operation involves using a message digest obtained via a secure hash function. A digest $h(M)$ of a message M is a short string of bits (for example, a 128-bit string) obtained by applying a hash function h to the message M.

 Published by the IEEE Computer Society 1089-7801/03/$17.00©2003 IEEE

Two desired properties of the hash function are

- *Collision-free*: different messages lead to different message digests, and
- *Noninvertible*: given a message digest, it is impossible — or difficult — to obtain the message.

Message digests are used in digital signatures, which I'll discuss further in the next section. Examples of secure hash functions, ranging from less to more secure and from faster to slower, are MD5, SHA, and SHA-1. Message-digest generation is a fast operation compared to public key encryption. SHA-1's hash-generation rate, for example, is 13 clock cycles per byte on a Pentium machine.[4] So, we could generate a digest of a 1-Mbyte file in approximately 13 milliseconds on a 1-GHz Pentium machine. Encrypting the same file with a 512-bit public key would take a few seconds on the same machine.

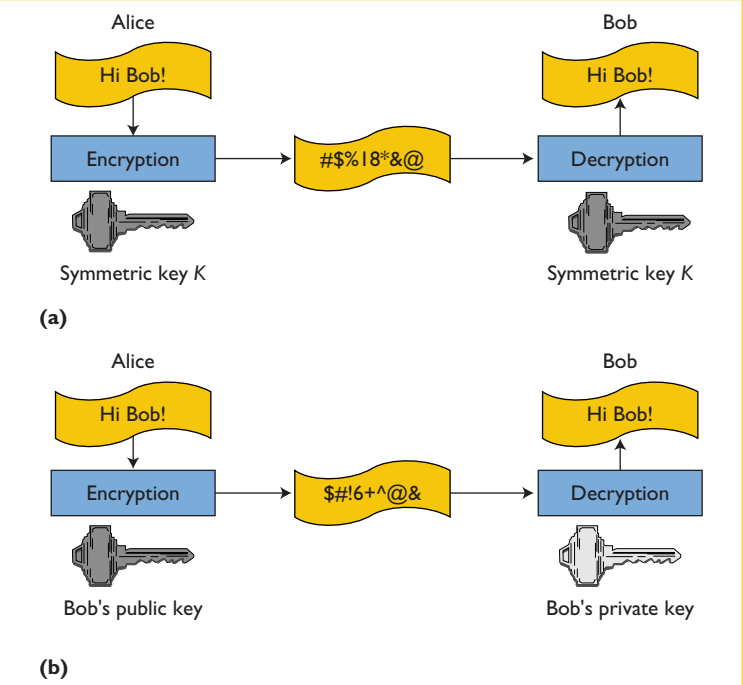

Figure 1. Symmetric key encryption versus public key encryption. (a) In symmetric encryption, Bob and Alice must share the same key, which is used for both encryption and decryption. (b) In public key cryptography, Alice encrypts the message with Bob's public key so that he can decrypt it with his private key.

Composite Security Operations

Composite security operations and protocols use the same basic cryptographic operations I just described as building blocks. The two most common composite operations are digital signatures and authentication using the Secure Sockets Layer (SSL) protocol.

Digital Signatures

A digital signature's purpose is to ensure integrity and nonrepudiation when files are transmitted. For example, proposals submitted to a funding agency or orders to trade stocks might have to be signed digitally. To do this, *A* must follow specific steps to send a digitally signed message *M* to *B*; *B* must follow additional steps to verify that *A* signed the message:

1. *A* creates a message digest $h(M)$ and encrypts it with its private key. The result is the digital signature *d* of *M*.
2. *A* sends (*M*, *d*) to *B*.
3. *B* decrypts *d* using *A*'s public key and gets $h(M)$. To do so, *B* must have previously obtained *A*'s public key, which is usually achieved by acquiring a digital certificate for *A* containing its public key. A trusted certificate authority must sign this certificate.
4. *B* generates a message digest $h(M)$ with the same hash function *A* used. If this digest matches the digest obtained in step 3, then the signature is verified. In other words, *B* knows that *A* generated the signature because *A*'s public key could decrypt the digest encrypted with *A*'s private key (assuming, of course, that only *A* knows its private key). *B* knows that message *M* was not modified because if it were, the message digest that *B* generated would not match the one that *A* sent.

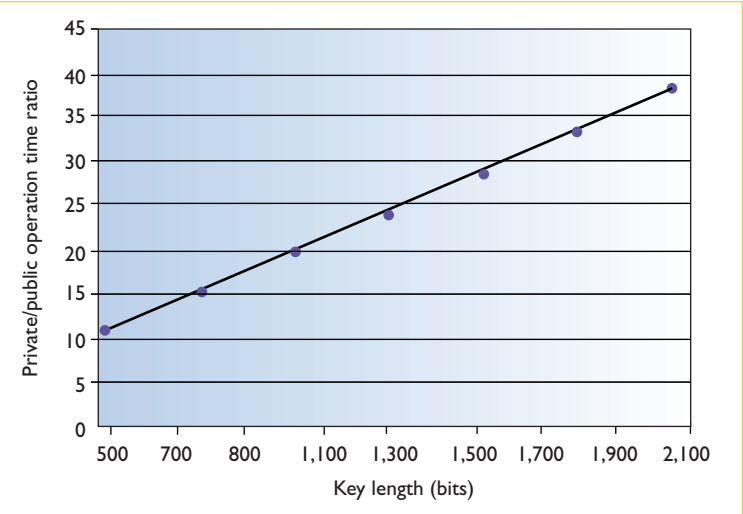

Figure 2. Ratio of private key over public key operation times versus key length. The ratio is almost linear but public and private key times grow with a power of the key length.

Thus, in a digital signature operation, signing a message requires one hash and one private key operation, and verifying a signature requires one

Table 1. Digital signature verification times and maximum throughput for two hash functions and various key sizes.

Hash function	Public key size (bits)	Verification time (seconds)	Maximum throughput (requests per second)
MD5	512	0.067	14.9
	768	0.068	14.8
	1,024	0.068	14.6
	2,048	0.073	13.8
SHA-1	512	0.093	10.8
	768	0.093	10.7
	1,024	0.094	10.6
	2,048	0.098	10.2

hash and one public key operation. Signing a message takes much longer than verifying the signature does. This property is useful because a server has to verify signatures generated by a large number of clients. (Many clients do the work of signing separately whereas a server has to verify the messages sent by all clients.)

Obviously, security performance trade-offs differ for various digital signature options. Suppose, for example, that a funding agency receives Web uploads of proposals that are, on average, 1.5 million bytes long. The agency's server must verify the digital signature associated with each proposal it receives. Table 1 shows the verification time for two hash functions, MD5 and SHA-1, and various key sizes used in public key decryption of the received message digest.[5] The verification time is the sum of the time needed to generate a message digest of the entire proposal and the time needed to decrypt, with a public key, the small encrypted digest received with the proposal. In this case, message-digest generation occupies the bulk of the verification time because the public key operation is applied to a small, fixed-size string of bits. The last column of Table 1 shows the site's maximum possible throughput, which is the inverse of the verification time.[5]

The rows in Table 1 appear in increasing order of security: SHA-1 is more secure than MD5, and the longer the key, the stronger the security. Verification time increases as security increases, but maximum throughput decreases with security. As the table shows, hash function choice has the biggest influence on performance. The verification time using SHA-1 is between 35 and 38 percent higher than for MD5 in this example.

Secure Sockets Layer

SSL is the most common authentication protocol used in e-commerce transactions. It handles mutual or one-way authentication and preserves the integrity and confidentiality of data exchange between clients and servers. SSL has two phases: a handshake phase and a data-transfer phase. During handshake, a server sends its public key, within a digital certificate,[5] to a client. The client generates a secret and sends it to the server, encrypted with the server's public key. The server decrypts the secret with its private key. At this point, both client and server can use the common secret to generate a key for symmetric encryption during data transfer.

This approach has the advantage of using symmetric encryption, which is much faster than public key encryption, for bulk data transfer. Public key encryption is used for information exchanged during the handshake.

Figure 3 shows an e-commerce site's maximum throughput under several scenarios.[5] In scenario A, all requests go over a nonsecure connection. Scenario B uses SSL with RC4 for symmetric key encryption, MD5 for the hash function, and a 512-bit key for public key encryption during handshake. Scenarios D and F are like scenario B except that the public key lengths are 768 and 1,024 bits, respectively. Scenarios C, E, and G use 3DES for the symmetric key encryption algorithm and SHA-1 for the secure hash function. The public key lengths in these scenarios are 512, 768, and 1,024 bits, respectively.

Figure 4 shows a partial order that indicates the security performance trade-offs of Figure 3's various scenarios. As Figure 4 shows, we cannot establish a relationship between some of the scenarios. Scenario F, for example, uses RC4 and MD5 during the data transfer phase, making it less secure but faster than scenario E, which uses a more secure and slower option — 3DES and SHA-1 — during the data transfer phase. However, scenario F uses a

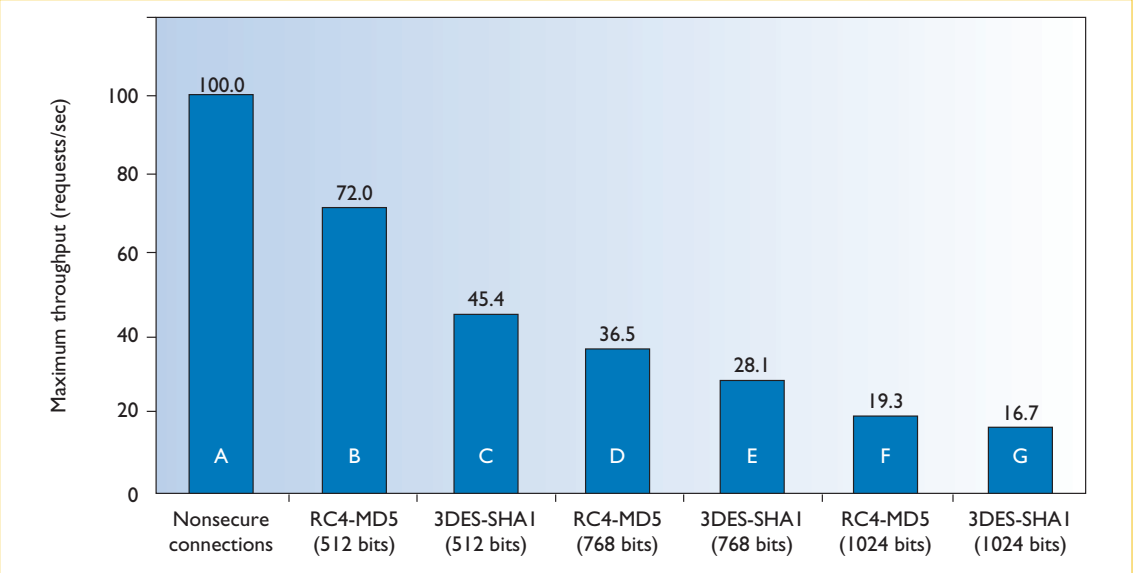

Figure 3. Maximum throughput of Secure Socket Layer for various combinations of cryptographic algorithms and key lengths. For the scenarios in this figure, we use two symmetric key algorithms (RC4 and 3DES), two hash functions (MD5 and SHA-1), and three key lengths (512, 768, and 1,024 bits) for the public key operations.

longer public key than scenario E does for the handshake, which makes scenario F more secure and slower for the handshake than scenario E.

Final Recommendations

The examples described here underscore the importance of understanding the level of security required for each application to fend off possible threats while minimizing performance penalties. Bruce Schneier indicates that by 2005, a public key of 1,280 bits should be sufficient to protect against attacks from individuals, but a 1,536-bit key will be needed to protect from attacks originated in large corporations. A 2,048-bit key would be the best protection against attacks from the government. Each of these selections carries a certain performance cost.[2] Therefore, a careful quantitative analysis of the performance impacts of security protocols must be combined with an analysis of potential threats so that good-enough security mechanisms are deployed for each situation.

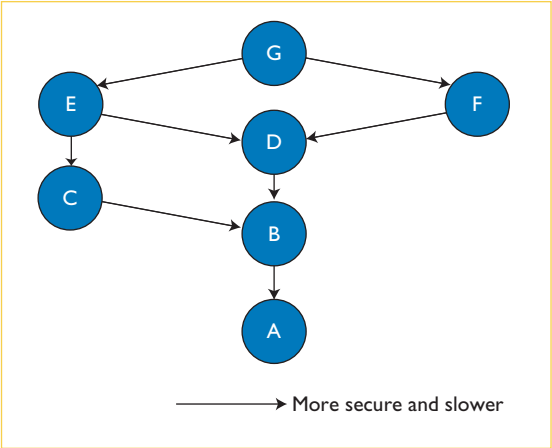

Figure 4. Security–performance relationships between Figure 3's scenarios. The arrows go from more-secure to less-secure options and from scenarios of worse to better performance.

References

1. R. Sandhu, "Good-Enough Security," *IEEE Internet Computing*, vol. 7, no. 1, 2003, pp. 66–68.
2. B. Schneier, *Applied Cryptography*, 2nd ed., John Wiley & Sons, 1996.
3. W. Freeman and E. Miller, "An Experimental Analysis of Cryptographic Overhead in Performance-Critical Systems," *Proc. 7th IEEE Int'l Symp. Modeling, Analysis, and Simulation of Computer and Telecomm. Systems* (MASCOTS'99), IEEE CS Press, 1999, pp. 348–357.
4. B. Schneier et al., "Performance Comparison of the AES Submissions," *Proc. 2nd AES Conf., Nat'l Inst. Standards and Technology*, 1999, pp. 15–34; www.counterpane.com/aes-performance.html.
5. D.A. Menascé and V.A.F. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*, Prentice Hall, 2000.

Daniel Menascé is a professor of computer science, the co-director of the E-center for E-Business, and the director of the MS in E-commerce program at George Mason University. He received a PhD in computer science from UCLA and authored the books *Capacity Planning for Web Services* and *Scaling for E-Business* (Prentice Hall, 2002 and 2000, respectively). He is a fellow of the ACM and a recipient of the A.A. Michelson Award from the Computer Measurement Group.