The Basis of HTTPS

David Giese

When you access your bank account online, your web browser communicates with the bank's server over a public network. The HTTPS protocol is designed to keep this communication private, but how is this possible when the entire process occurs over a public channel? How do you ensure that you are communicating with the bank's server and not a computer that is masquerading as the bank? Before discussing the cryptography behind HTTPS, I will first present a quick review of hash functions.

A *hash function* is a function that deterministically maps an arbitrarily long piece of data to a fixed-length piece of data. The fixed-length piece of data is called a *digest* or a *hash.* Hash functions are by definition not one-to-one; various inputs will share the same digest such that the original data cannot be unambiguously derived from the digest. Hash functions used in cryptography must have the property that, given a digest, it is difficult to find other messages that produce the same digest.

An *encryption procedure* transforms a message into an encrypted form, which is unreadable until it is *decrypted.* An encryption (or decryption) procedure consists of a general method and a key. In *symmetric cryptography* the encryption key and the decryption key are the same, while in *asymmetric cryptography* the two keys are different and each key is able to decrypt messages that were encrypted by the other.

Symmetric cryptography cannot provide secure communication over a public channel unless the key is exchanged using a private channel. Typical internet applications have no practically available private channel, hence symmetric cryptography is unappealing. Asymmetric cryptography, which was invented at the GCHQ in 1973, provides a solution to this problem.

Its use is best illustrated with an example: Imagine that David wishes to send a private message to Blayne. To do this, David asks Blayne to produce a pair of keys and to give him one of the keys (his *public key*), while keeping the other key (his *private key*) secret. David then encrypts his message with Blayne's public key, and sends it to him. Finally, Blayne decrypts the message with his private key. Blayne can similarly send a message to David, so that there are four keys involved in the procedure.

This simple approach is insecure as is because a third party may intercept the public keys and send fake messages. To fix this, a *digital signature* is included with all encrypted messages. A digital signature is a digest of the message that is encrypted with *the sender's* private key. The recipient then decrypts the message with their private key, and decrypts the digest with the sender's public key. They can then verify that the message is intact and not sent by a third party by calculating the digest of the message and comparing it to the one sent in the digital signature.

Digital signatures still do not ensure that the original recipient is who they claim to be; Blayne may actually be Daniel. To protect against this, public keys are registered by a trusted third party, called a *certificate authority*, and the certificate authority's public keys are known *a priori.* David can then privately and securely ask for Blayne's public key from the certificate authority, instead of asking for it directly, thus ensuring that Blayne is not Daniel.