cryptographic-protocols-between-arduino-and-PC

Arpit Chauhan, Inderjit Sidhu and Archit Pandey

Communication on networks is vulnerable to eavesdropping and other malicious activities. Depending on the sensitivity of the information being accessed, the communicating entities need to ensure that unauthorized entities cannot access the information (**confidentiality**), an unauthorized entity cannot modify the message (**integrity)**, the service is not denied to authorized users (**availability**), the message really originated from the claimed user (**authenticity**), and each message can be uniquely traced to an entity (**accountability**). In order to fulfill these requirements, various cryptographic algorithms and protocols have been devised. The prominent ones have been standardized by authorities like NIST (National Institute of Science and Technology), and ISO (International Organization for Standardization).

The project aims at demonstrating how two entities communicating on a channel may go about implementing such protocols and algorithms. The algorithms that this project entails are the Advanced Encryption Standard (AES), RSA[1], Diffie-Hellman key exchange[2] and MD5. AES is the "gold standard" in symmetric cryptography. NIST evaluated fifteen competing designs for the standard, before selecting the Rijndael cipher as the most suitable and announcing AES as FIPS PUB 197[3]. AES was also included in the ISO/IEC 18033-3[4] standard. The high speed and low requirements of the cipher makes it feasible to implement it on systems with very low configurations, e.g. an 8-bit microcontroller. On the other hand, RSA is a widely accepted algorithm for asymmetric cryptography.

Encryption is the process of encoding messages (or information) in such a way that eavesdroppers or crackers cannot read it, but that authorized parties can.
In an encryption scheme, the message or information, known as plaintext is encrypted using an encryption algorithm into ciphertext. The process usually involves one or more encryption keys. An algorithm is suitable only if it is *computationally infeasible* for an adversary to get to know the plaintext without the key.

In a symmetric encryption model, a message X is encrypted with a key K to generate the ciphertext, i.e.

$$Y = E(K, X)$$

The message can be recovered with the same key K:

$$X = D(K, Y)$$

---

[1] Rivest, R. L., Shamir, A., & Aldeman, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. Retrieved from http://people.csail.mit.edu/rivest/Rsapaper.pdf

[2] Diffie, W., & Hellman, M. (1976).
Retrieved from http://www-ee.stanford.edu/~hellman/publications/24.pdf

[3] Federal Information Processing Standards Publication 197
Retrieved from http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[4] ISO/IEC 18033-3:2010 Standard
Retrieved from http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54531

On the other hand, in an asymmetric cipher model, there is a pair of keys, one called the public key PU and the other the private key PR. A message X encrypted with one of the keys can be decrypted only by using the other key. That is,

$$Y = E(PU, X) \; and \; X = D(PR, Y)$$

or $$Y = E(PR, X) \; and \; X = D(PU, Y)$$

These algorithms (AES and RSA) are used in various cryptographic protocols. These protocols may incorporate key agreement or establishment, entity authentication, digital signature, and secure digital time-stamping etc. In this project, only the following protocols have been implemented:

(1) Symmetric encryption key exchange using a public-key algorithm, and subsequent transfer of data encrypted with the exchanged key
(2) Digital Signature using a cryptographic hash function, and a public-key algorithm

The protocol (1) will be implemented with two different sets of algorithms. In one implementation, symmetric encryption will be done using AES-128 and public-key algorithm will be RSA. In the second implementation, symmetric encryption will be done again using AES-128 and public-key algorithm will be Diffie-Hellman.

In protocol (2), SHA-256 will be used as the cryptographic hash function and RSA will be the public-key algorithm.

The entities will be called Alice (or A) and Bob (or B) in accordance with the common convention in cryptology literature.

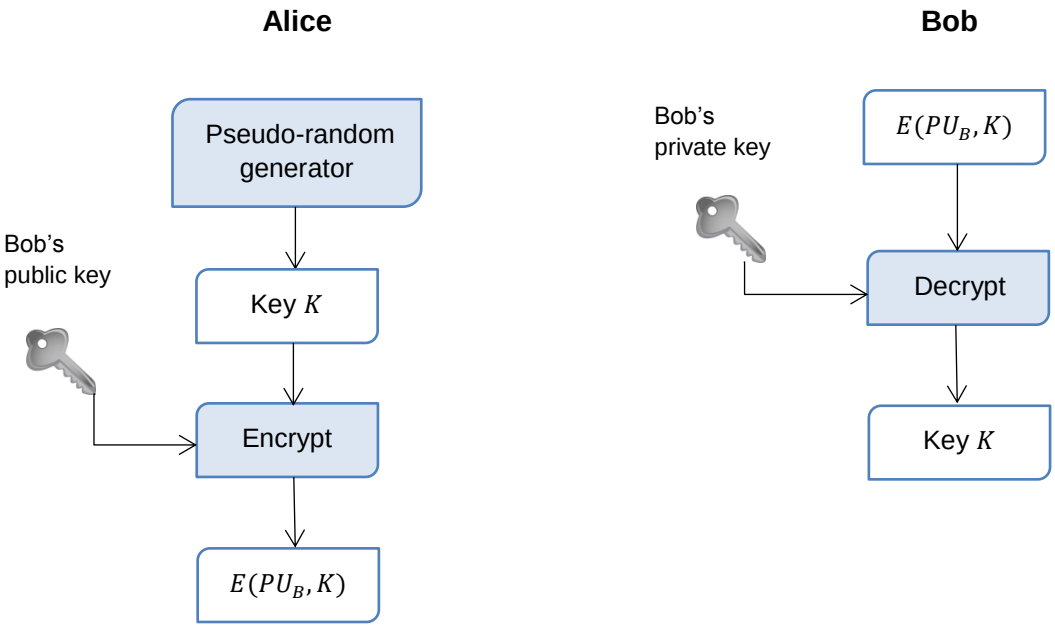
Key exchange

The problem with symmetric cryptography is that both the entities have to know the key in order to be able to read the information. This problem is not present when using asymmetric cryptographic algorithms, as the entities have a pre-defined mechanism to obtain each other's public keys. Even then, all the information needed to be transmitted isn't encrypted using asymmetric algorithms because they are slow. Instead, they are used just to transmit the key for symmetric algorithms in a secure manner.

The key exchange might be done by either using RSA or Diffie-Hellman algorithm.

(a) RSA:

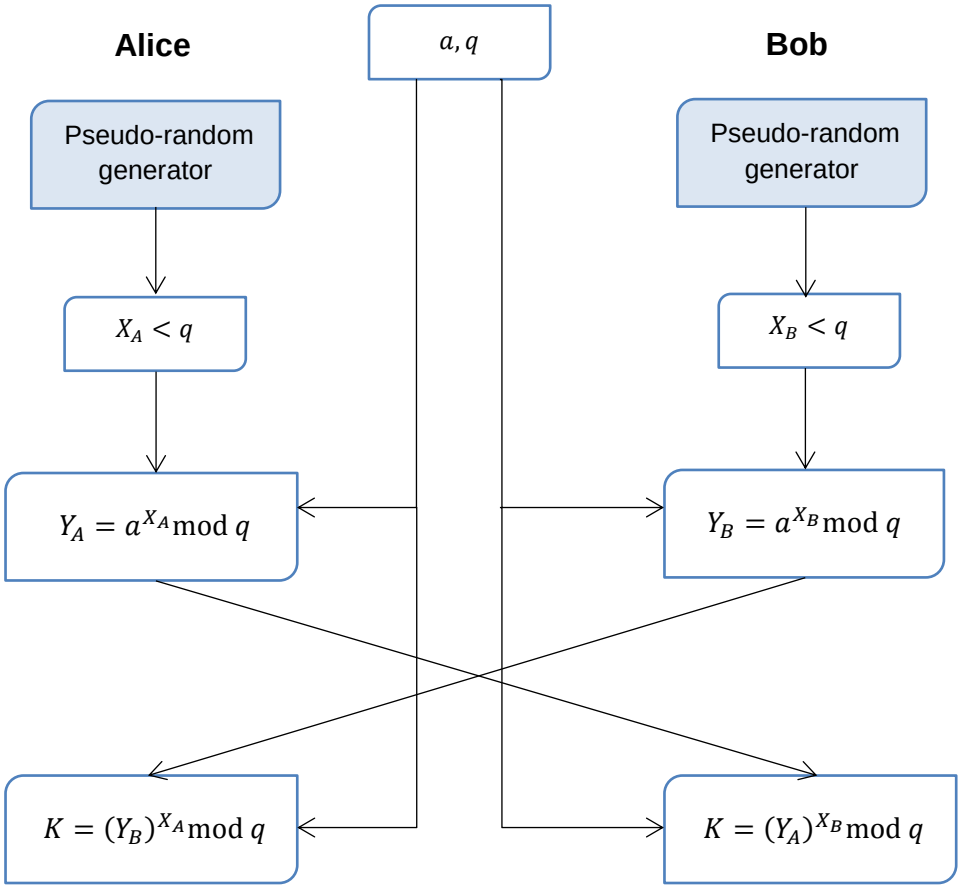
The entities Alice and Bob know each other's public keys. The entity willing to initiate the communication (let's say Alice) generates a pseudo-random 128 bits key K, encrypts it using Bob's public key and sends it. Bob receives the key in encrypted form and decrypts it using its own private key. Since only Bob knows its private key, only it can get to know the key. Now, both the entities share a common key K, which they can subsequently use for transmitting information.



(b) Diffie-Hellman Key Exchange:

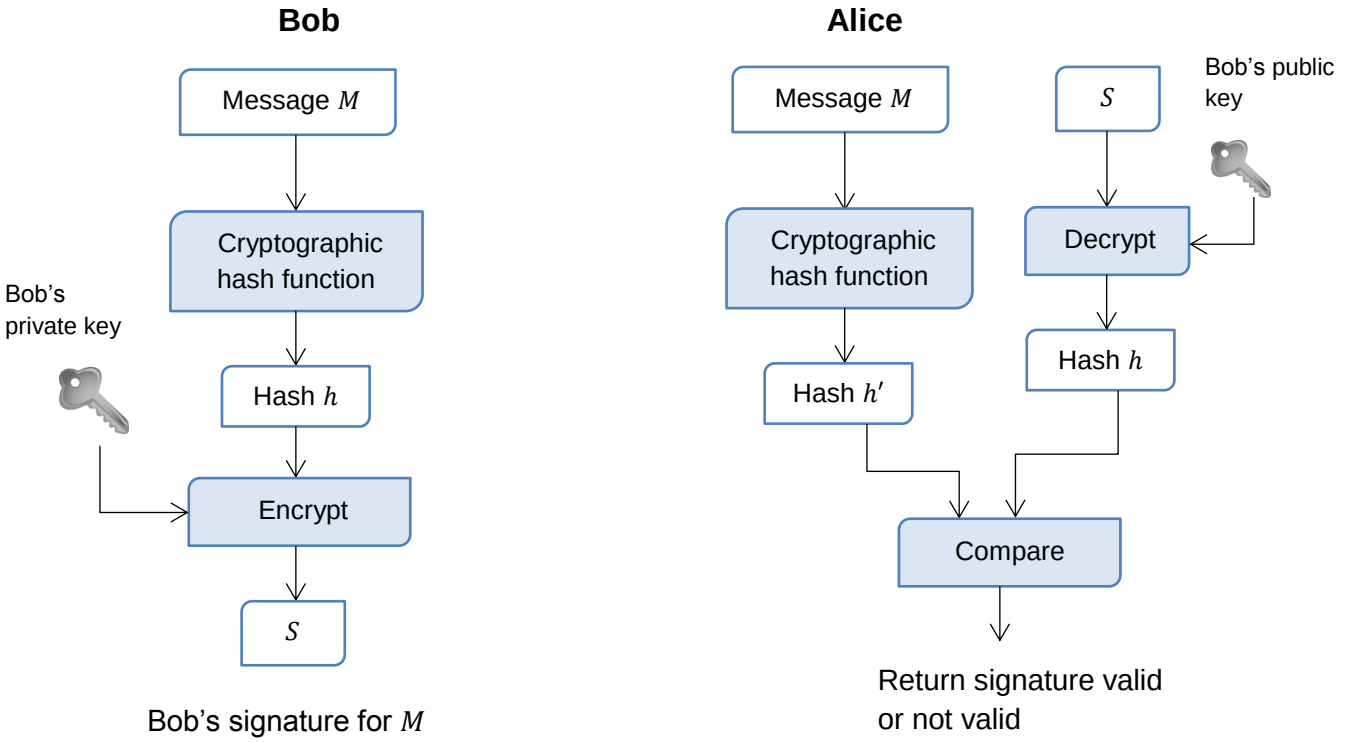
The entities Alice and Bob know two numbers: a prime number q and an integer a that is a primitive root of q. The entity Alice selects a random integer $X_A < q$ and computes $Y_A = a^{X_A} \bmod q$. Similarly, the entity Bob selects a random integer $X_B < q$ and computes $Y_B = a^{X_B} \bmod q$. Each entity keeps the X value private and makes available the Y value to the other entity. Alice computes the key as $K = (Y_B)^{X_A} \bmod q$ and Bob computes the key as $K = (Y_A)^{X_B} \bmod q$ (the two calculations provide identical results).

$$
\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (a^{X_B} \bmod q)^{X_A} \bmod q \\
&= (a^{X_B})^{X_A} \bmod q, && \text{by the rules of modular arithmetic} \\
&= a^{X_B X_A} \bmod q \\
&= (a^{X_A})^{X_B} \bmod q \\
&= (a^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}
$$



Digital Signature

Suppose the entity Bob wants to sign a message M. It passes M through a cryptographic hash function, to produce the hash h. Subsequently, Bob encrypts h using its private key (RSA). The resultant S is Bob's signature for the message M. Bob encrypts the message M and the signature S using the common AES-128 key K, and then sends it to Alice. To verify that the signature S was done on the message M only, Alice passes M through the same cryptographic hash function to obtain the hash h' and obtains the hash h by decrypting S using Bob's public key. If $h' = h$, Alice concludes that the message M was signed by Bob.



Cryptographic Hash Functions

A hash function H accepts a variable-length block of data M as input and produces a fixed-size hash value $h = H(M)$. A good hash function when applied to a large set of inputs will produce outputs that are evenly distributed and apparently random. The principal object of a hash function is data integrity. A change to any bit or bits in M results, with a high probability, in a change to the hash value.

A cryptographic hash function is expected to be

(i) **one-way**, i.e. it must be computationally infeasible to find a data object that maps to a pre-specified hash value

(ii) **collision-free**, i.e. it must be computationally infeasible to find two data objects that map to the same hash value

The MD5 Message-Digest algorithm is a widely used cryptographic hash function. It is specified in RFC 1321.[5] It has been utilized in a wide variety of security applications, but it has been shown that it is not collision-resistant. [6] Thus, its use is not recommended anymore and it has been succeeded by algorithms like SHA-1 and SHA-2.

SHA-1 is a hash function created by the National Security Agency (NSA) of the United States. It was issued as FIPS 180-1[7] by NIST and also specified in RFC 3174[8]. It produces a hash value of 160 bits. In 2002, NIST produced a revised version of the standard, FIPS 180-2, that defined three new versions of SHA, with hash value lengths of 256, 384 and 512 bits, known as SHA-256, SHA-384, and SHA-512, respectively. Collectively, the hash algorithms are known as SHA-2. A revised document FIPS 180-3 was issued in 2008, adding a 224-bit version.

Although, SHA-2 family is more secure, in this project, MD5 will be used because it's easier to implement.

Pseudo-random number generators (PRNGs)

PRNGs are used in a variety of cryptographic applications, e.g. for generating keys for symmetric encryption or generating public and private keys for RSA, generating nonces and salts. Two criteria must be met by the sequence generated by a PRNG: **uniform distribution** (the distribution of bits in the sequence should be uniform) and **independence** (no one subsequence in the sequence should be inferable from the others). In this project, we will use the pseudo-random number generators available in the commonly available libraries.

In this project, two independent devices function as two entities interested in communication on a channel. The devices used are an Arduino microcontroller and a Windows-based PC. Communication is established using a USB serial adapter that serves as a virtual COM port for the PC. The devices implement the cryptographic protocols discussed above.

The project is hosted at https://github.com/arpitchauhan/cryptographic-protocols-arduino-and-PC

---

[5] Request for Comments 1321
Retrieved from http://tools.ietf.org/html/rfc1321

[6] Wang, X., & Yu, H. (n.d.). Retrieved from http://merlot.usc.edu/csac-f06/papers/Wang05a.pdf

[7] Federal Information Processing Standards Publication 180-1
Retrieved from http://www.itl.nist.gov/fipspubs/fip180-1.htm

[8] Request for Comments 3174, Internet Engineering Task Force (IETF)
Retrieved from http://www.ietf.org/rfc/rfc3174.txt