An efficient implementation of Diffie-Hellman key exchange protocol on UDOO

Sachin Rathod and Sahaj Biyani
{rathod,sahajbiyani}@cs.ucsb.edu
How can two parties agree on a secret value when all of their messages might be overheard by an eavesdropper?

The Diffie-Hellman [1] key agreement protocol (1976) was the first practical method for establishing a shared secret over an unsecured communication channel.

The point is to agree on a key that two parties can use for a symmetric encryption, in such a way that an eavesdropper cannot obtain the key.

The Diffie-Hellman algorithm accomplishes this, and is still widely used.
Diffie-Hellman Algorithm Analogy

Alice

Common paint

+ 

Secret colours

= 

Bob

Common paint

+ 

Secret colours

= 

Public transport

(assume that mixture separation is expensive)

Alice’s secret colours

+ 

Bob’s secret colours

= 

Common secret
Steps in the Algorithm:

1. Alice and Bob agree on a prime number $p$ and a base $g$.
2. Alice chooses a secret number $a$, and sends Bob $(g^a \mod p)$
3. Bob chooses a secret number $b$, and sends Alice $(g^b \mod p)$
4. Alice computes $((g^b \mod p)^a \mod p)$
5. Bob computes $((g^a \mod p)^b \mod p)$
We tried different exponentiation methods to compute the key values to compare their performance on different platforms.

Three methods of exponentiation:

1. Binary Exponentiation (Implemented)
2. Montgomery Exponentiation (Implemented)
3. OpenSSL (Used from library)
For managing arbitrary length numbers, we used OpenSSL’s BIGNUM structure [2] and its library functions.

This library performs operations on integers of arbitrary size. The operations include arithmetic (add, multiply etc.), comparison, conversion to different formats etc.
One of the methods we used for analysis is binary exponentiation. The binary exponentiation method is explained by the following algorithm:

**Input:** $M, e, n$.

**Output:** $C = M^e \mod n$.

**Step 1.** 
if $e_{k-1} = 1$ then $C = M$ else $C = 1$

**Step 2.** 
if $i = k - 2$ downto 0
2a. 
$C = C \cdot C \mod n$
2b. 
if $e_i = 1$ then $C = C \cdot M \mod n$

**Step 3.** return $C$
Another method we used for analysis is Montgomery exponentiation. The Montgomery exponentiation method is explained by the following algorithm:

\textbf{function} \ MonPro(\bar{a}, \bar{b}) \\
\text{Step 1.} \quad t = \bar{a} \cdot \bar{b} \\
\text{Step 2.} \quad m = t \cdot n' \mod r \\
\text{Step 3.} \quad u = (t + m \cdot n)/r \\
\text{Step 4.} \quad \text{if } u \geq n \text{ then return } u - n \\
\quad \quad \text{else return } u
Montgomery Exponentiation Method

function ModExp(M, e, n) { n is odd }
Step 1. Compute \( n' \) using Euclid’s algorithm
Step 2. \( \bar{M} = M \cdot r \mod n \)
Step 3. \( \bar{C} = 1 \cdot r \mod n \)
Step 4. for \( i = k - 1 \) down to 0 do
Step 5. \( \bar{C} = \text{MonPro}(\bar{C}, \bar{C}) \)
Step 6. if \( e_i = 1 \) then \( \bar{C} = \text{MonPro}(\bar{M}, \bar{C}) \)
Step 7. \( C = \text{MonPro}(\bar{C}, 1) \)
Step 8. return \( C \)
Results are compared between UDOO board and standard PC with following configurations:

<table>
<thead>
<tr>
<th>UDOO</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1 x [ARMv7 Processor rev 10 (v7l)]</td>
</tr>
<tr>
<td>Physical Memory</td>
<td>800 MB</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 12.04 32-bit</td>
</tr>
</tbody>
</table>
Cryptographic Engineering
Implementation of D-H Key Exchange on UDOO

Diffie-Hellman Parameters

- Prime $p$ and generator $g$:
  1. IETF standard 1024 and 2048-bit primes and corresponding generators (having 160-bit and 224-bit prime order subgroups). RFC5114 [3]
  2. Random 'safe' primes generated using OpenSSL library having given number of bits and generator $g$ is taken as 5). (Safe primes are of the form $2p + 1$, where $p$ is also prime)

- Safe primes are of the form $2p + 1$, where $p$ is also prime. Safe primes offers security against Pohlig and Hellman attacks, but require more computation.

- Parameters $a$ and $b$ : random primes with given number of bits
Avg time required for key generation on UDOO (in seconds):

<table>
<thead>
<tr>
<th>Key-size (bits)</th>
<th>Binary Exponentiation</th>
<th>Montgomery Exponentiation</th>
<th>OpenSSL Exponentiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>0.005414833</td>
<td>0.009804000</td>
<td>0.001707833</td>
</tr>
<tr>
<td>512</td>
<td>0.023968332</td>
<td>0.047772333</td>
<td>0.008993666</td>
</tr>
<tr>
<td>1024</td>
<td>0.148043826</td>
<td>0.284063160</td>
<td>0.058445834</td>
</tr>
<tr>
<td>2048</td>
<td>0.294208169</td>
<td>0.564812660</td>
<td>0.114655666</td>
</tr>
</tbody>
</table>
## Comparing UDOO and PC

**Avg time required for key generation (in seconds):**

<table>
<thead>
<tr>
<th>Key-size (bits)</th>
<th>Binary Exponentiation</th>
<th>Montgomery Exponentiation</th>
<th>OpenSSL Exponentiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024 [UDO0]</td>
<td>0.148043826</td>
<td>0.284063160</td>
<td>0.058445834</td>
</tr>
<tr>
<td>1024 [PC]</td>
<td>0.007844172</td>
<td>0.018422132</td>
<td>0.001439296</td>
</tr>
<tr>
<td>2048 [UDO0]</td>
<td>0.294208169</td>
<td>0.564812660</td>
<td>0.114655666</td>
</tr>
<tr>
<td>2048 [PC]</td>
<td>0.015397863</td>
<td>0.036434080</td>
<td>0.002855158</td>
</tr>
</tbody>
</table>
Conclusions

D-H key generation performance:

- Binary exponentiation 2-3 times faster than Montgomery exponentiation.
- OpenSSL implementation of exponentiation is 3 times faster than our binary exponentiation.
- This could be because OpenSSL implementation is highly efficient than our implementation.
Key Learnings from the Project

We learnt a lot from this project. Some of the learnings are as follows:

- Hands-on development on the UDOO platform.
- The use of OpenSSL library for handling arbitrary length integer operations in C programming language.
- The implementation of security protocols and operations in secure and efficient manner.
Future Work

Future iterations of this project can include:

- Improving efficiency of Montgomery exponentiation implementation for UDOO board.
- Using the key exchange implementation to communicate messages between remote clients and testing its security.
References

