Security Applications

Abhyudaya Chodisetti
Paul Wang Lee
Garrett Smith

1. Introduction

Cryptography applications generally involve a large amount of processing. Thus, there is the possibility that these applications would present a good target for CMP's. In this survey, we examined the characteristics of major cryptography applications to determine whether this is the case.

The major components of cryptography include encryption, MAC's, digital signatures, and authentication. Encryption hides the content of a message from eavesdroppers, providing privacy. MAC's provide a way to detect tampering, thereby providing message integrity. Digital signatures enable a party to verify the origin of a message. Authentication provides a way to identify another party. Most extended applications such as electronic cash, electronic voting, secure auctions, etc. use these or some extension of these major components.

These components, in turn, are built using a few basic functions, such as block ciphers, secure hash functions, and public key encryption. For example, digital signatures use secure hash functions and public key encryption, and MAC's are created using secure hash functions. Block cipher or public key encryption is used directly for encryption.

Therefore, examining these three key functions will enable us to gain an understanding of the properties and requirements of most cryptography applications. (There is another class of encryption methods, stream ciphers, but it was omitted from our study as they seem to be less interesting. Stream ciphers generally involve generation of pseudorandom numbers from a seed and XORing to the message to be encrypted.)

2. Block Ciphers

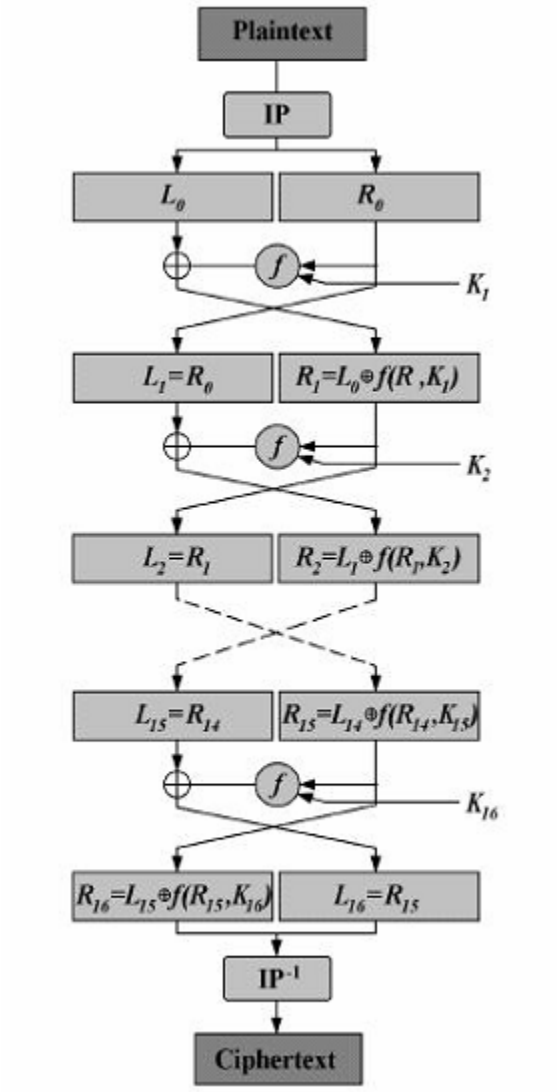
Block ciphers work on a block of plain text (typically 64-256 bits) as a whole and produce a cipher text block of equal length. We examined two block ciphers in particular, DES and AES.

DES

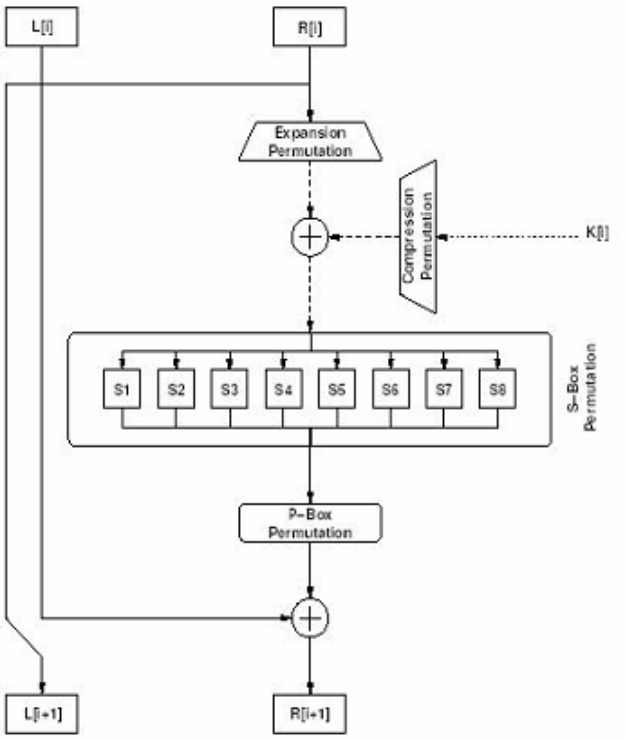
DES(Data Encryption Standard) is a block cipher that has been in use since the 1970's.

DES operates on blocks of 64 bits each and uses a key size of 56 bits. It is composed of 16 rounds, each one performing the same operations, but using different inputs. The inputs to a given round are the two 32 bit outputs from the previous round (L[i] and R[i]) and 48 bits of data derived from the key (K[i]). Half of the input data at each round is first permuted/expanded and XORed with a round key, and then divided into 8 6-bit chunks. A 'substitution box' (S-box) is used to convert this 6 bit value to a 4 bit value, which is then concatenated with all the other outputs of the S-boxes. This value is then XORed with the other half of the input data, producing the inputs to the next round. S-boxes can be

implemented efficiently in hardware using four 6-input single-output gates. In software, they are implemented as lookup tables.



Global Structure of DES



Structure of one DES round

Because of the design of the round structure, the computation of a DES encryption / decryption is highly serialized. The S-box lookups are independent, but since they are implemented as small lookup tables, they are all memory accesses. Another property of DES is its bit permutations, which are hard to implement in software.

DES's 56 bit key is too short to be safe from brute force attacks given today's computers, so a variant, Triple DES is used, encrypting each block 3 times with different keys. This further increases the amount of computation per block.

AES

AES is the new federal standard that replaces DES. Its original name is Rijndael. It can operate on blocks of 128, 192 and 256 bits and can use keys of 128, 192 and 256 bits. It also uses the round structure, but the content of each round is different from DES. In each round 4 operations are performed:

1. Byte Substitution: A byte substitution is performed, based on the results of a table lookup. The table is generated from a Galois Field, and provides the non-linearity in the system.
2. Shift Rows: Each row is shifted by different byte amounts.
3. Mix Column: Mix the bytes within a column in a linear way, using table lookups and arithmetic.
4. Key Addition: XOR in new key material.

AES is easier to implement in software, due to its avoidance of bit permutations.

An important issue in using block ciphers is the mode of their operation when encrypting messages of general length. Two representative modes are ECB(electronic code book) mode and CBC(cipher block chaining) mode. The important difference between these two modes is the way they encrypt successive blocks. In ECB, every block of the message is encrypted independently of the previous block. In CBC mode, each block's encryption depends on the encryption of the previous block. ECB mode exposes information about the original message, and is thus insecure. Hence, CBC mode is more commonly used, but this introduces serial dependency limiting the available parallelism that can be exploited.

Properties of DES/AES

Both DES and AES rely heavily on table lookups, with AES relying even more heavily. Because of this there are two different data access patterns to look at: The table access patterns and the data access patterns.
The tables are relatively small (2K for DES 1K for AES). These are accessed randomly, based on the input data, but are small enough to fit fully in L1 cache, and in hardware implementations can be implemented directly with gates. The key (56bits for DES, 128-256 bits for AES) is small enough to fit in registers.
The input data can be of arbitrary size, depending on the application that is using cryptography. But it is operated on serially and in very small chunks at a time (64 bits for DES, 128-265 bits for AES), which fit in the register file. These algorithms operate in a streaming manner over the input data, with very predictable memory access patterns, thus prefetching would be very effective.
Encryption in general is O(N) with N being the amount of data to be encrypted.

Software Implementations

We investigated the NIST's published reference implementation of AES and the implementation of DES that came with OpenSSL 0.9.7b. To determine the ratio of ALU operation to memory references we profiled test programs to locate the inner loop for both AES and DES, and then compiled the relevant function to assembly and analyzed the output by hand. The target architecture for all tests was SPARC v8.

For DES, the encryption of a block of data takes 192 memory references and 520 other instructions. This is a ratio of 2.7 ALU ops per memory reference. Of those 192 references 128 are table lookups. For AES, one round of an encryption takes about 111 memory references, and 507 other instructions. This is a ratio of 4.56 operations per memory reference. AES used more loops in its main encryption function, which we weren't able to analyze completely, so these numbers are more approximate. AES has a decent amount of ILP within a round, due to the fact that it operates on larger blocks, but both DES and AES are not exceptional in the amount of parallelism, with a current processor being able to take advantage of most of the parallelism available. (Clapp).

3. Hashing Functions

SHA-1 is a secure hash function, which means that given any message, it is very hard to devise another message that has the same hash. This is used to prevent tampering and as a building block for digital signatures and MAC's. Block ciphers can be used as hash functions, but they are slower than algorithms designed explicitly for hashing. SHA-1 is specially designed for hashing, and is one of the most popular secure hash functions.

The structure of SHA-1 is similar to DES and AES. It uses multiple rounds, but it doesn't need incorporate any nonlinear transforms, unlike block ciphers, so it can avoid table lookups. It operates on a message digest of 160 bits and 512 bits of message text at a time. At the end of processing 512 bits of message text, it outputs a 160 bit message digest which is used as the input for the next block. While processing one 512 bit block of message text it performs 80 rounds of computation, incorporating some of the message text in each round.

We investigated SHA-1 in the same way that we did AES and DES. We found that in 80 rounds it did 232 memory operations and 2879 other operations. This results in 12 ALU operations per memory reference. This much higher number is due to the lack of table lookups. The amount of ILP appears to be reasonable, but nothing exceptional.

4. RSA

RSA is the most widely used public key encryption algorithm. It relies on mathematical properties of modular arithmetic to provide its security properties.

For each user, there are two keys, a public key and a private key. Each of these keys is a pair of numbers: (e, M), (d, M) where M is the modulus for all arithmetic. A message, a value between 0 and M, is encrypted by taking its modular power of the number e in the public key. To decrypt the resulting message, the d'th modular power is taken. The resulting value is the original value that was encrypted.

If P is the message(plaintext) to be encrypted, D is the encrypted message(ciphertext), and the two keys are (e, M), (d, M), then encryption and decryption are as follows.

- Encryption: $C = P^e \bmod M$
- Decryption: $P = C^d \bmod M$

This relationship arises from the fact that d, e, and M are chosen to exploit a certain property in modular arithmetic.

RSA Properties

The recommended size for e, d, and M are 2048 bit integers, although e and d are usually values that are relatively small. The message is the same size as M, since it is a value between 0 and M. The encryption and decryption algorithms involve modular exponentiation. With an efficient implementation, the complexity scales as $O(k^3)$, where k is the number of bits of the key.

Using the same methodology as for the previous algorithms we looked at the RSA implementation that came with OpenSSL. Since modular multiplication is the core of RSA we looked at that function and found that there was a ratio of 1:1 for ALU operations to memory operations, which is due to fact the multiplications were being done on 2048 bit numbers, and so each chunk of the number had to be brought in from memory, worked on and written out.

5. General comments

The fundamentally serial nature of many cryptography algorithms mean that single process thread level parallelism is rare. The numerous permutations and non-linear transformations also tend to hide much of data level parallelism.

An interesting attribute of many cryptography algorithms is that the operations to be performed on input data is completely deterministic. Thus the load does not depend at all on the input data set, other than input length.

6. Future trends

Cryptography applications have a very small number of conservative standards. Correctness is very difficult to prove and any vulnerability can lead to catastrophic results. Because of these factors, rate of change is relatively slow. For example, AES is targeted for use in the next 20 years. Given this, it makes sense to embed special purpose hardware if performance is important.

Future scaling of requirements will mostly arise from higher number of independent requests per time, and for block ciphers, some message length scaling. The former will provide more opportunities for multiprocessors in the same way as network applications do, but there will be no gain in ILP or DLP.

7. Custom Hardware

Some algorithms such as DES were originally implemented in hardware, and DES ASIC blocks are widely incorporated into network or security chips.
Commercially available cryptography coprocessors include hardware implementations of DES/AES and modular arithmetic modules for RSA along with a simple RISC core. Some examples are the CS5210/40 family of AES encryption cores which achieve upto 25.6Gbits/s encryption, Broadcom's cryptographic and security processors targeted towards high bandwidth and e-commerce such as BCM5801, BCM5802.

Somewhat related are secure coprocesors. They are tamper proof sealed devices that have a processor, memory storage and sometimes fast crypto support. They are protected in that any attempt to penetrate them will result in all critical memory being erased. They can thus store secure cryptographic keys and are used for E-commerce applications such as digital cash, copyright protection etc.
Some examples are mu-ABYSS and Citadel systems from IBM, iPower encryption card by National Semiconductor.

8. Benchmarks

The benchmarks for symmetric encryption and secure hashes all involve running the algorithm over a large amount of data and calculating the megabytes/gigabytes per second that the implementation can process data. Since asymmetric algorithms operate on fixed size amounts of data, their speed is usually measured in encryptions per second. As long as the implementation produces the proper output, it is legal.

9. Conclusion

Special purpose hardware is a good idea for cryptography applications. The slow change of encryption standards and the simplicity of the algorithms make hardware implementations of block ciphers, secure hash functions and modular multiplication a sensible choice for any application that expects to use encryption.
The major scaling direction for cryptographic applications will be independent thread level. There is very little interesting parallelism available within the encryption of one stream of data, and current processors are able to take advantage of most of the parallelism at that level. However the number of streams that must be handled concurrently most likely increase, with encrypted network connections being the most obvious domain where this will happen.

References

1. Stallings, William. *Cryptography and Network Security: principles and practice.* Upper Saddle River, NJ: Prentice Hall, 1999.
2. Stinson, Douglas. *Cryptography : Theory and Practice, 2nd Edition*
3. *DES Encryption Standard*. Federal Information Processing Standards Publication 46-2
4. Daemen, Joan & Rijmen, Vincent *AES Proposal: Rijndael*. Document version 2, March 1999.
5. RSA Laboratories, *RSA Cryptography Standard*. Public-Key Cryptography Standards #1
6. Clbapp, Craig S.K. *Instruction-level Parallelism in AES Candidates.* Second AES Candidates Conference, Rame, March 1999.
7. N. Weaver, J. Wawrzynek, *High Performance, Compact AES Implementation in Xilinx FPGAs*.
8. R. Lee et al. *Efficient Permutation Instructions for Fast Software Cryptography*, IEEE Micro, Nov. 2001.
9. B. Yee and J. D. Tygar, *Secure Coprocessors in Electronic Commerce Applications*, USENIX95, 1995. 14
10. E. Mosanya, C. Teuscher, H. F. Restrepo, P. Galley, and E. Sanchez. *CryptoBooster: A Reconfigurable and Modular Crytographic Coprocessor*. In . Kog and C. Paar, editors,

Proceedings: Workshop on Cryptographic Hardware and Embedded Systems, volume LNCS 1717, pages 246-256, Worcester, Massachusetts, USA, August 1999. Springer-Verlag