Session Keys for Encryption/Decryption in Elliptic Curve Cryptosystems

Donovan Moore, Michael Gubody, and Tai-Chi Lee
Department of Computer Science and Information Systems
Saginaw Valley State University
University Center, MI 48710

Abstract - *This work evolved from a case study on an Elliptic Curve Cryptosystem (ECC) [2], where the session keys are used with FPGAs in the process of encryptions or decryptions [4]. To improve the strength of encryption and the speed of processing, the public key and the private key of ECC are used in 3BC (Block Byte Bit Cipher) [1, 5, 11] algorithm, which generates session keys for the data encryption. We are investigating a novel approach of hardware co-design implemented in VHSIC Hardware Description Language (VHDL), which produces hardware algorithm for heavy iterations to be placed onto the FPGAs, thereby gaining a speed-up by a subroutine call to a sequence of custom instructions executed on the FPGAs.*

Keywords: EEC, FPGA, VHDL, 3BC algorithm

1 Introduction

The session keys play an important role in the process of encryptions/decryptions for an ECC (Elliptic Curve Cryptosystem). The majority of products that use public-key cryptography for encryption/decryption use RSA algorithm. But as we know, the key length for secure RSA has increased over the years. This would demand a heavy computing power for applications, especially for electronic commerce site that process a large number of transaction. Recently, a different approach of generating public key based on elliptic curve cryptography (ECC) has begun to challenge the weakness of RSA [12]. Its security relies on the problem of computing logarithms on the points of an elliptic curve. The main attraction of ECC is that it appears to offer equal security for a far smaller key size, thereby saving the processing overhead. To improve the strength of encryption and the speed of processing, the public key and the private key of ECC are used in the 3BC (Block Byte Bit Cipher) algorithm, which generates session keys for the data encryption. Fundamentally, ECC (Elliptic Curve Cryptosystem) technique is more mathematics involved. We only give a brief review of the basic concept in the next section and will explain elliptic curve ciphers later.

2 EC (Elliptic Curves) - Mathematical Overview

The elliptic curve cryptosystem makes use of elliptic curve in which the variables and coefficients are all restricted to elements of a finite field. Two families of elliptic curves are used in cryptographic applications. They are prime curves defined over Zp and binary curves constructed over GF(2^n). In general, cubic equations for elliptic curve over real numbers takes the form

$$y^2 + axy + by = x^3 + c\,x^2 + d\,x + e. \quad (1)$$

where a, b, c, d, e are real numbers that satisfy some conditions. For our purpose, we will limit to the case where a, b, and c are zero, which results in the form

$$y^2 = x^3 + d\,x + e. \quad (2)$$

To plot such curve, we need to compute

$$y = (x3 + d\,x + e)^{1/2} \quad (3)$$

For given values of a and b, the plot consists of positive and negative values of y for each value of x. Thus each curve is symmetric about y = 0. Figure show two examples of elliptic curves. In the definition of an elliptic curve, we include a single point O called a point at infinity or the zero point, and also if any three points on EC lie on a straight line, their sum is O. Thus, the addition of two points on EC is defined as follows:

1) O serves as the additive identity so that
O + O = O, and
P + O = P, for any P ε EC

2)There exists an inverse for any point P on EC. The inverse of P = (x,y) on EC is
-P = (x,-y), and the subtraction is defined as
Follows P - Q = P + (-Q), for any P, Q ε EC

3)An associative law.
P + (Q + P) = (P + Q) + R, for any P, Q, R ε EC

4)A commutative law.

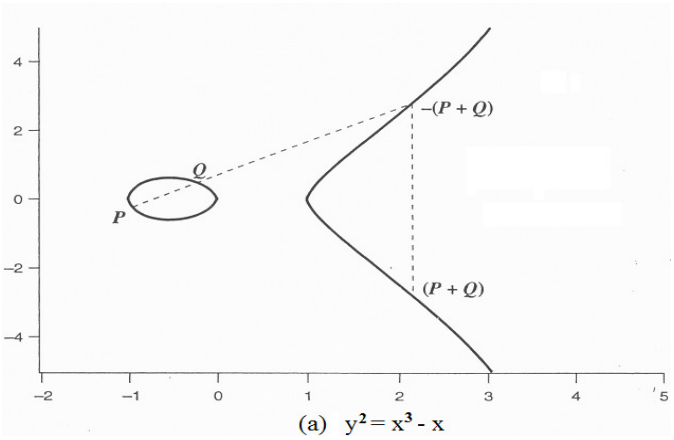
$P + Q = Q + P$, for any P, Q ε EC

For any two distinct points P = (xp, yp), Q = (xq, yq) that are not negative of each other, the slope of the line *l* that joins them is m = (yq - yp) / (xq - xp). There is exactly one other point where *l* intersects the elliptic curve, and that is the negative of the sum of and Q. After some algebraic manipulation, we have the sum R = P + Q as follows:

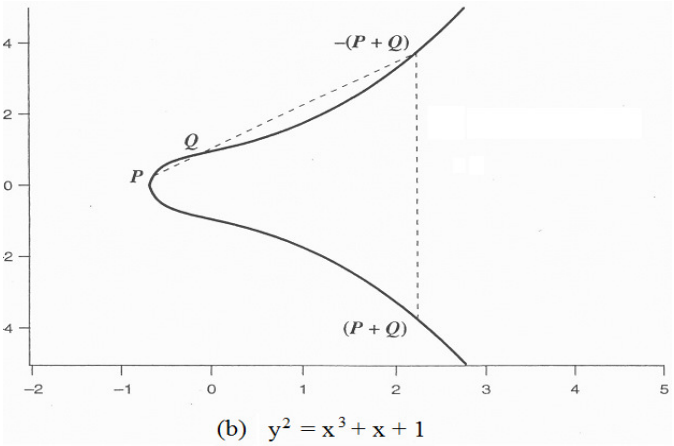
$$x_r = m^2 - xp - xq$$
$$y_r = - yp + m (xp - xr)$$

We also need to be able to add a point to itself, that is P + P = 2P = R. When $yp \neq 0$, we have

$$xr = [(3xp^2 + a)/2yp]^2 - 2xp$$
$$yr = [(3xp^2 + a)/2yp] (xp - xr) - yp$$



(a) $y^2 = x^3 - x$



(b) $y^2 = x^3 + x + 1$

Figure 1 . Examples of Elliptic Curves

2.1 ECC (Elliptic Curve Cryptosystem)

The concept of ECC, which was proposed by N. Kobiltz [5] and V. Miller [11] in 1985 is that when any two points are selected and added, the point of the sum is generated and is used for cryptosystem. The elliptic curve (EC) over real numbers x is the set of points (x,y) to satisfy a equation y2 = x3 + ax + b. If the right side term x3 + ax + b doesn't have multiple root, that is, 4a3 + 27b2 ≠ 0, EC gives us some geometric features to work with. To apply EC to cryptosystem, the computation with modulo p is used where p is prime number. As a result of this calculation, rounding errors to be raised by real number computation can be prevented. The procedure to generate a public key in ECC is outlined as follows:

(1) [Sender] Select any prime number p
(2) [Sender] Select any integer number a, b for EC such that $y^2 = x^3 + ax + b$
(3) [Sender] Select randomly an initial point P on EC
(4) [Sender] Generates a random integer as private key K_S
(5) [Sender] Computes a public key K_S by multiplying P by K_S and registers it in the public key directory.
(6) [Sender] Transmits p, a, b, P, K_SP to Receiver
(7) [Receiver] Receives p, a, b, P, K_SP from Sender
(8) [Receiver] Generates a random integer K_R as a private key
(9) [Receiver] Computes a public key K_RP by multiplying P by K_R and registers it in the public key directory.

Note: It is easy to verify that $K_S K_R P = K_R K_S P$

3 Encryption and Decryption Algorithm

As shown in Figure 2, the user A computes a new key $k_A(k_BP)$ by multiplying the user B's public key by the user A's private key k_A. The user A encodes the message by using this key and then transmits this cipher text to user B. After receiving this cipher text, The user B decodes with the key $k_B(k_AP)$, which is obtained by multiplying the user A's public key, k_AP by the user B's private key, k_B. Therefore, as $k_A(k_BP) = k_B(k_AP)$, we may use these keys for the encryption and the decryption.

Select factor: a, b, $y^2 = x^3 + a x + b$.
p: prime number, P: Point on EC

User A	User B
Private key: K_A	Private key: K_B
Public key: K_AP	Public key:K_BP
en/decryption key: $K_A(K_BP)$	en/decryption key: $K_B(K_AP)$

Figure 2. Concept of en/decryption of ECC

In the next section, we investigate 3BC cipher algorithm, which is a different approach that adds an additional level of complexity to enhance the security of encryption.

3.1 Encryption and Decryption with 3BC Algorithm

With 3BC algorithm, the procedure of data encryption is divided into three parts, inputting plaintext into data block, byte-exchange between blocks, and bit-wise XOR operations between data and session key.

3.2 Session Key Generation

As we know that the value which is obtained by multiplying one's private key by the other's public key is the same as what is computed by multiplying one's public key to the other's private key. The feature of EC is known to be almost impossible to estimate a private and a public key. With this advantage and the homogeneity of the result of operations, the proposed 3BC algorithm uses a 64-bit session key to perform the encryption and decryption. Given the sender's private key Ks and the receiver's public key Pr, we multiply Pr by Ks to obtain a point KsPr = (X, Y) on EC, where X= X1 X2,…Xm and Y= Y1 Y2,…Yn. Then we form a key N by concatenating X and Y (i.e. N = X1 X2,…Xm Y1 Y2,…Yn), and generate the session keys as follows:

i) If the length (number of digits) of X or Y exceed four, then the extra digits on the left are truncated. And if the length of X or Y less than four, then they are padded with 0's on the right. This creates a number N' = X1' X2' X3' X4' Y1' Y2' Y3' Y4'. Then a new number N'' is generated by taking the modulus of each digit in N' with 8.

ii) The first session key sk1 is computed by taking bit-wise OR operation on N" with the reverse string of N".

iii) The second session key sk2 is generated by taking a circular right shift of sk1 by one bit. And repeat this operation to generate all the subsequent session keys needed until the encryption is completed. For more details on the use of public key and session key for encryption and decryption process, see [6].

3.3 Block Data Input

The block size is defined as 64 bytes. A block consists of 56 bytes for input data, 4 byte for the data block number, and 4 byte for the byte-exchange block number (see Figure 5). During the encryption, input data stream are blocked by 56 bytes. If the entire input data is less than 56 bytes, the remaining data area in the block is padded with each byte by a random character. Also, in the case where the total number of data blocks filled is odd, then additional block(s) will be added to make it even, and each of those will be filled with each byte by a random character as well. Also, a data block number in sequence is assigned and followed by a byte-exchange block number, which is either 1 or 2.

3.4 Byte Exchanges between Blocks

After filling the data into the blocks, we begin the encryption by staring with the first data block and select a block, which has the same byte-exchange block number for the byte exchange. In order to determine which byte in a block should be exchanged, we calculate its row-column position as follows:

For the two blocks whose block exchange number, n = 1, and given the values of a sender's public key 21135 and a receiver's private key 790, the row and column can be computed.

For n = 1, It follows from 3.2 that N" = 11357900 (after truncation, padding and concatenation), and

row = ((1,1,3,5,7,9,0,0)*1) mod 8 = (1,1,3,5,7,1,0,0) and

col = (((1,1,3,5,7,9,0,0)*1+3) mod 8) = (4,4,6,0,2,4,3,3)

This results 8 byte-exchange positions, (1,4), (1,4), (3,6), (5,0), (7,2), (1,4), (0,3) and (0,3). However, counting only once for repeating pairs, the four bytes at (1,4) (3,6), (5,0), and (7,2) will be selected for byte-exchange between two blocks.

For block exchange number n = 2, we have

row = ((1,1,3,5,7,1,0,0)*2) mod 8 = (2,2,6,2,6,2,0,0) and

col = (((1,1,3,5,7,1,0,0)*2+3) mod 8 = (5,5,1,5,1,5,3,3),

which results 8 byte-exchange positions, (2,5), (2,5), (6,1), (2,5), (6,1), (2,5), (0,3) and (0,3). Similarly, only three byte positions at (2,5), (6,1), and (0,3) are used for byte-exchanges between two blocks.

3.5 Bitwise XOR between Data and Session Keys

After the byte-exchange is done, the encryption proceeds with a bit-wise XOR operation on the first 8 byte data with the session sk1 and repeats the operation on every 8 bytes of the remaining data with the subsequent session keys until the data block is finished.

Note that the process of byte-exchange hides the meaning of 56 byte data, and the exchange of the data block number hides the order of data block, which needs to be assembled later on. In addition, the bit-wise XOR operation transforms a character into a meaningless one, which adds another level of complexity to deter the network hackers.

4 Theoretical Implementation

To generate a public key, the most time consuming process is to find an initial point P on the given elliptic curve and to compute kP for an integer k < p for a large prime number p. The approach we investigate in this paper is to create a 64bit ALU with its own custom instructions added to an Altera EP2C35 NIOS II embedded processor. Custom instructions are designed to be small, rearrangeable portions of a C implementation of key generation. This will allow sections of the algorithm to be in C and other sections to be expressed as custom instructions. These sections can be easily reordered and re-factored by recompiling the algorithm and uploading the overlay to the FPGA via TCP/IP in order to handle the distribution of the algorithms over the network.

4.1 Hardware Design

The design of this approach consists of four components: A PC Master Controller, TCP/IP interconnect, FGPA logic units that each contains a NIOS II processor and custom ALU, and the creation and selection of the custom instructions and overlays.

4.2 PC Master Controller

A PC Master Controller will provide benefits over existing designs. It is capable of systematically assigning algorithms to logic units based on the specific set of custom instructions included in the ALU. Our design will implement the delegation of operations and also take advantage of the parallelism that can be obtained by using a FPGA [3,7,8,10].

4.3 TCP/IP Interconnect

The PC Master Controller will communicate through a TCP/IP interface with one or more FGPAs in a cluster. Each FPGA will execute the algorithm, using the custom instructions.

4.4 Custom Instructions

The Altera EP2C35 NIOS II embedded processor is a 32bit system. By adding a customized 64bit ALU and associated 64bit registers [9, 12, 13], we can have custom instructions to handle algorithms specific to public key via EC, which include: XOR, Addition, Multiplication, Division, Right/Left Shift, and others. These instructions are given 32bit UUIDs as their opcode, allowing unique naming even when the full set is not within a single ALU. We are experimenting with various decompositions of Expansion and Permutation of the 3BC algorithm between PC Master Controller and custom instructions for optimal results.

5 Current Implementation

5.1 Implementation of 3BC Encryption

The current approach to implementing the 3BC encryption algorithm assigns the blocking and byte exchange to the PC Master Controller, with the bitwise right rotation and XOR handled on the ALU implemented on an FPGA. The current model involves using the ALU to generate one 64-bit session key from a prior session key to encrypt 64-bits of data.

5.2 Hardware Implementation

At the time of writing, the hardware model is as follows:

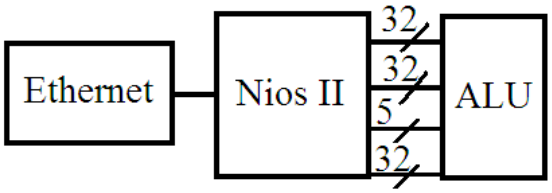


Figure 3. Current Implementation Model with Custom ALU

In figure 3, two 32-bit input buses provide key, plaintext, and the number of bits to shift the prior session key into the ALU for key generation, a 5-bit control bus for the NIOS II custom instruction, and one 32-bit output bus for the encrypted result from the ALU. The NIOS II chip handles the network connection and manages the barrel shift and XOR by means of a custom instruction with some I/O handling for data locality.

The inside of the ALU is outlined below:

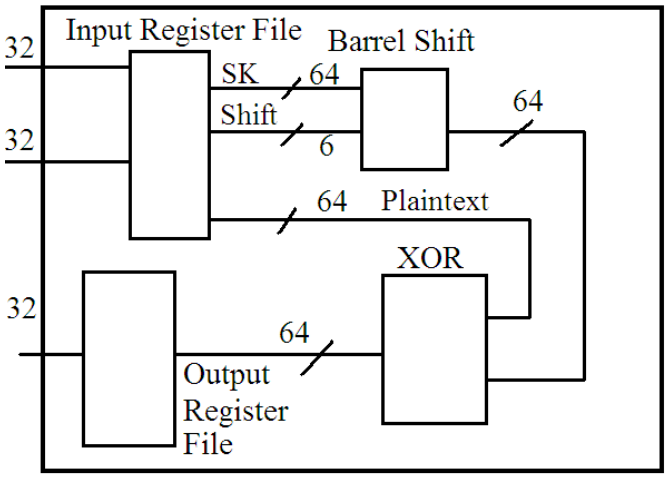


Figure 4. Inside of Custom ALU on FPGA
(Not pictured: 5-bit control bus for custom instruction)

This model suffers from the limitation of requiring 3 clock cycles to input the data from the NIOS II chip to the input

register file, 2 clock cycles to output the data, an assumed 1 clock cycle to process the data, as well as any inherent delays from Ethernet data transfer. Future research plans include changing to 64-bit data buses as input to the custom ALU, or a change to a different System on a Programmable Chip (SOPC) processor. With these, the input delay can be reduced to 2 clock cycles or even 1. A 64-bit data bus for output from the ALU allows for output to be condensed into 1 clock cycle, allowing for the process to be completed in an expected 3-4 clock cycles.

6 Acknowledgements

The student authors would like to acknowledge the SRCI mini-grant foundation for the opportunities it provides, SVSU for providing a prosperous research environment, and Dr. Lee for his resources and encouragements.

7 Empirical Results

Different decompositions of the 3BC sub-algorithms are being investigated. With the approach on the encryption/decryption algorithm outlined in Section 4, the expectation is to process on average one key block per clock cycle, excluding filling and extracting the custom ALU data. This appears reasonable as the custom instructions allow the design to use several FPGAs to process multiple key ranges simultaneously. Of interest is the location of the balance between the high implementation time with the low run time of the pure hardware approach, and the low implementation time with high run time of the pure software approach.

For instance, starting with a pre-developed encryption or decryption algorithm in C that has a nonexistent design time, its average run time is a constant. When the algorithm is translated into well optimized hardware, the design time is very high and the run time is very low. With the approach outlined in Section 4, the design time and run times are between the pure hardware and pure software methods. When the number of data sets to run is in the bolded range on Figure 3, this method should be preferable.

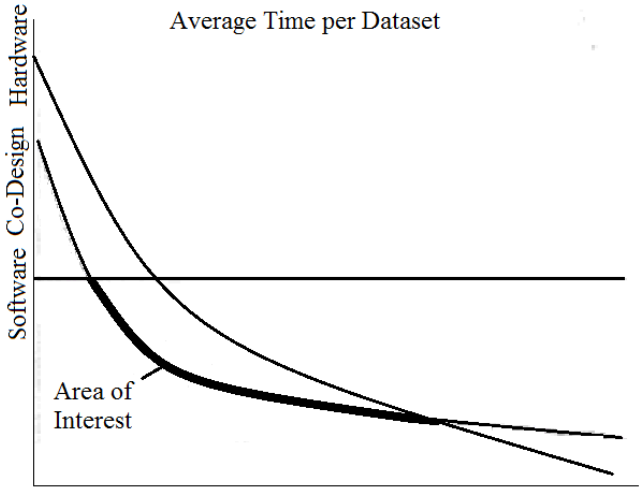


Figure 5. Area of Interest with Co-Design

7.1 Difference in Efficiency

The parallel algorithm described in section 4 allows for a large increase in speed over the pure software approach.

For pure software, the assumption is that the first session key (SK1) has already been generated, there are 6 additional clock cycles needed to generate the session keys for the first block (SK2-7), and 7 clock cycles are needed to XOR the data with session keys. Thus, a total of 14 clock cycles are required for each block after the first. For n blocks, the operations take f(n) = 14n clock cycles. This is an operation in O(n) time.

With sufficient hardware, the data can be input in 1 clock cycle. After the first block, the algorithm requires 7 operations that can be performed in parallel for session key generation. With the FPGA, this can be accomplished in 1 clock cycle. There are then 7 XOR operations between session keys and data that can also be performed in parallel. This, too, with the FPGA can be performed within 1 clock cycle. Lastly, an anticipated 1 clock cycle for gathering the results and output. Thus, for n blocks and sufficient hardware, the operations take f(n) = 4 clock cycles. This is an operation in O(1) time.

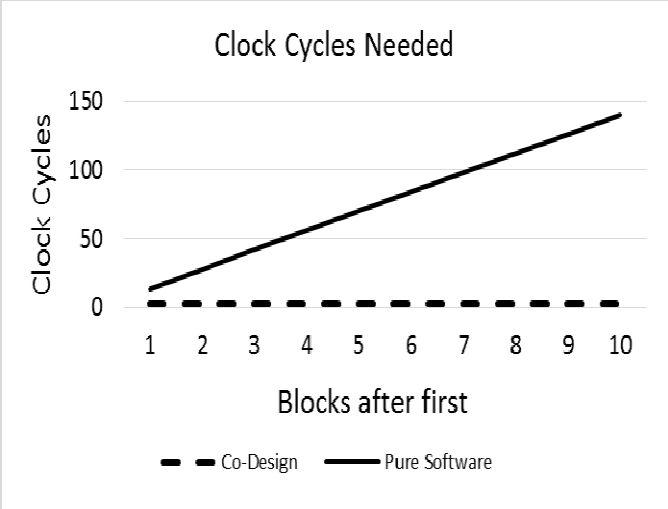


Figure 6. Theoretical Efficiency gained with sufficient hardware

8 Conclusion

The researched approach has the benefit of some of the speed of ASIC, while maintaining some of the flexibility of C. The added use of storing and transferring the algorithms as an overlay allows the organizational aspects of the algorithm to be re-factored and delivered without the need of rebuilding the ASIC image and reconfiguring the FPGA [10] Using a TCP/IP interconnect network to send both the overlay and the problem set allows for an efficient and easily scalable infrastructure.

The proposed 3BC, which uses byte-exchange and the bit operation increases data encryption speed. Even though cipher text is intercepted during transmission over the network. Because during the encryption process, the 3BC algorithm performs byte exchange between blocks, and then the plaintext is encoded through bit-wise XOR operation, it rarely has a possibility for cipher text to be decoded and has no problem to preserve a private key

9 References

1. M.J. Bastiaans, FPGA's as Cryptanalytic Tools, http://www.sps.ele.tue.nl/
members/m.j.bastiaans/spc/rouvroy.pdf.
2. A. Fernaades, Elliptic Curve Cryptography, Dr. Dobb's Journal, December, 1999.
3. P. Glesner, and M. Zipf, Renovell (Eds.): Programmable Logic and Applications. Reconfigurable Computing Is Going Mainstream, Proceedings of 12th International Conference, FPL 2002, Montpellier, France, September 2-4, 2002.
4. Jilani Ibrahim, Li Lan, Shi Yixin, VHDL implementation of Data Encryption Standard (DES), pp. 3-6, 16, ECE Department, University of Illinois at Chicago May 9th, 2003.
5. N. Koblitz, Elliptic Curve Cryptosystems. Math. Comp. 48 203-209, 1987.
6. Tai-Chi Lee, Byung Kwan Lee, An ASEP (Advanced Secure Electronic Payment) Protocol Design, The Proceedings of The IEEE 2004 International Conference e-Technology, e-Commerce and e-Service, Taipei, Taiwan, pp. 41-46, 3/28-31, 2004.
7. Tai-Chi Lee, Mark White, Using Software Emulation in FPGAs To Improve Co-Design Development Time, The Proceedings of the Fourth International Conference of Applied Mathematics and Computing, Bulgaria, pp. 359-360, August 12-18, 2007.
8. Tai-Chi Lee, Patrick Robinson, A FPGA-Based Designed for an Image Compressor, International Journal of Pure and Applied Math, Academic Publications, Volume 33 No.1, pp. 63-67, 2006.
9. Tai-Chi Lee, Richard Zeien, Adam Roach, and Patrick Robinson, DES Decoding Using FPGA and Custom Instructions, The Proceedings of The Third International Conference on Information Technology: New Generation, Las Vegas, Nevada, pp. 575-577, 2006.
10. Tai-Chi Lee, Patrick Robinson, and Erik Henne, Framework for executing VHDL code on FPGA, The Proceedings of the International MultiConference in Computer Science & Computer, Las Vegas, NV, pp. 1296-1299, 2004.
11. V.S. Miller, Use of Elliptic Curve in Cryptography. Advances in Cryptology-Proceedings of Crypto '85, Lecture Notes in Computer Science 218, pp. 417-426, Springer-Verlag, 1986.
12. M. Robshaw, Block Ciphers, RSA Laboratories Technical Report TR, 601, August 1995, http://www.rsasecurity.com/rsalabs/dindex.html.
13. B. Schneier, Description of a New Variable-Length Key, 64-bit Block Cipher (Blowfish), Proceedings, Workshop on Fast 78 Software Encryption, New York: Springer-Verlag, 1993.