Customizing Programmable Hardware for Encryption Keys

Ingrid Biehl
(Techn. Universität Darmstadt, Germany
biehl@informatik.tu-darmstadt.de)

Andreas Grävinghoff
(FernUniversität-GH Hagen, Germany
andreas.graevinghoff@fernuni-hagen.de)

Jörg Keller
(FernUniversität-GH Hagen, Germany
joerg.keller@fernuni-hagen.de)

Abstract: Programmable hardware in embedded systems often faces price constraints because of the targeted market. But encryption in embedded systems demands powerful hardware which is expensive. Programmable hardware amortizes this investment because it can be used for other functions as well, but is slower than special purpose hardware. To relieve the tension on this side we propose acceleration of encryption on programmable hardware by customizing the circuits used. Instead of using a generic encryption circuit with an encryption key as input, we optimize the circuit for this key. We discuss a system how to generate and provide optimized circuit descriptions for different keys. We investigate popular encryption algorithms (DES, IDEA, RSA) and several AES candidates with respect to the acceleration due to this customization.

Key Words: encryption, block ciphers, reconfigurable/programmable hardware, optimization of encryption circuits
Category: B.2, B.7, C.3, E.3

1 Introduction

Embedded systems increasingly face a need to communicate data with other systems over long distances. This often necessitates the use of cryptography to protect the transmitted data, which in turn demands more processing power of the embedded system. On the other hand, embedded systems often face hard constraints in price (and thus performance) because of the targeted market. Using programmable hardware in embedded systems is gaining in popularity because its price can be amortized over several functions and because changes in functionality, e.g. a change of the encryption algorithm used, are simplified. These arguments also led to the development of so called custom computing machines which basically consist of a microprocessor and one or several programmable hardware modules, see e.g. [Hauser and Wawrzynek (97)].

In terms of performance however, programmable hardware is typically slower than special purpose hardware, which might hinder its use for the encryption of data to be transmitted with a reasonable bandwidth, especially if real-time requirements are to be fulfilled.

Our aim is to show that this disadvantage can be at least partially eliminated if the encryption algorithm used is optimized for the current key (called

customized in the sequel) and thus is faster on the same hardware platform. This is possible because the functionality of the programmable hardware can be changed each time a new key is used.

In [Section 2] we detail the basic idea of optimizing an encryption algorithm for a key, and discuss organisations for key directories that allow for the provision of optimized circuit descriptions to the embedded system. In [Section 3] we investigate several encryption algorithms with respect to the acceleration that can be expected by customizing them. In [Section 4] we summarize and give a brief outlook on further applications of customized encryption.

2 Customization of Encryption Algorithms

Cryptographic algorithms as for encryption, decryption, signing and signature-verification of data usually form a sequence of arithmetic and logical operations and are applied to a pair consisting of some data and a key. If we assume the key to be fixed, then operations working on a key are applied to a constant value. Suppose, for example, that parts of the data are multiplied with parts of the key. Then, this operation is multiplication with a constant if the key is fixed and can be implemented more efficiently in hardware than in case the key is variable. In [Section 3.2] we investigate the gain possible from this customization. Note that a major manufacturer of programmable hardware describes in an application note how to efficiently implement multiplication with a constant [Chapman (96)], but only mentions signal processing applications.

A simple variant of our idea can be found in [Schneier et al. (98)], where an option "compiled" for the Twofish algorithm is described. There, subkeys are treated as constants "...and directly embedded in a key-specific copy of the code...". Their variant allows to apply some code optimization, whereas customizing a multiplier circuit to a constant can result in acceleration by a factor of two. Also they give no rationale how to work in practice with fixed keys.

In general, customized cryptographic algorithms can be used in all kinds of applications which use cryptography. They just replace bare keys by descriptions of the corresponding customized algorithms. E.g. in our model sender and receiver of encrypted data use customized encryption and decryption algorithms instead of key-parameterised generic algorithms together with the appropriate keys as arguments. Correspondingly, key directories contain customized encryption and signature verification algorithms instead of public keys for public key encryption or digital signature schemes. Since optimization of the encryption resp. verification algorithm for each individual public key has to be done only once, even time-consuming optimization techniques may be acceptable for the computation of the customized algorithm, which is stored in the directory afterwards.

In the sequel, we will concentrate on implementations of encryption algorithms in hardware. In [Section 4], we will also give some ideas about the potential of our idea for software implementations. Depending on the description used for customized circuits they may be of larger size than the bare keys for which they are optimized. If storage or bandwidth for transmission of keys resp. customized descriptions is limited the following variants for the storage management of customized algorithms are conceivable. In case the optimization

process is not time-consuming only the key has to be stored together with a general optimization procedure and the customized algorithm is computed locally on demand. If a lot of keys resp. customized algorithms have to be stored and their key-dependent parts are small it is sufficient to store these key-dependent parts and to complete them as soon as needed [see Section 3.4].

As an example, consider a hybrid encryption scheme. The sender first generates a session key, which is communicated to the receiver by means of a public key encryption system. For the communication itself, a symmetric scheme is used. For the public key system, we can apply our idea or we can use a generic, i.e. not customized, public-key encryption algorithm if speed is not an issue here. To apply our idea to the symmetric encryption which is to follow we have several possibilities. The sender can compute the description of an encryption circuit customized to the session key, and send this description to the receiver. If the description is too large but its computation reasonably fast, then the session key itself is transmitted and the receiver computes the description locally. If the online computation of an encryption circuit customized to the session key would take too much time, then an optimization using a largely generic circuit can be used. In this case, only the description of the customized part is computed and transmitted.

3 Applicability

We investigate for several encryption algorithms whether customization results in an accelerated encryption hardware. For the promising among them we try to estimate the amount of acceleration quantitatively. We selected the algorithms on the basis of their importance as "standards" (sometimes de facto) in the field. We restrict to brief sketches of the algorithms themselves, for detailed descriptions see e.g. [Schneier (95)]. Information about candidate algorithms for the Advances Encryption Standard (AES) can be found at [NIST (00)].

3.1 DES

DES (Data Encryption Standard) is the standard among symmetric cryptographic algorithms. In recent times it shows signs of age because of its short keysize (64 bits of which 56 are used). There have been several successful attacks against DES (see e.g. RSA's competition, http://www.rsa.com and EFF's DES Cracker [EFF (98)].) More and more, DES is replaced by newer algorithms such as IDEA which we investigate next. Also noteworthy is the effort to find a successor called AES (Advanced Encryption Standard) [NIST (00)], of which we investigate some candidates.

DES consists of 16 identical rounds. In each round, key and message are processed separately. A round key is computed and combined with the intermediate result of the previous rounds by a bitwise exclusive OR. For a fixed key, the computation of the round keys can be omitted. Because of the bitwise exclusive OR however, the input to the next round still depends on the message, and no further optimizations are possible.

3.2 IDEA

IDEA uses messages and keys of 128 bit each, which are split into eight 16-bit blocks each. In 8 identical rounds, these blocks are combined by additions, multiplications, and bitwise exclusive or operations. In each multiplication and in half of the additions, the inputs are one block of the message and one block of the key. In these cases, a constant is added or multiplied which results in a reduction of depth and cost of the appropriate circuits. Hence, IDEA is well-suited for the proposed architecture.

To analyze the amount of acceleration, we restrict to the analysis of multiplication circuits because they dominate the encryption circuit's depth. The most common forms of multiplication circuits are array multipliers and Wallace-tree multipliers. When multiplying an integer a with another integer

$$b = \sum_{i=0}^{n-1} b_i \cdot 2^i \ ,$$

an array multiplier forms all partial products $a \cdot b_i$ and sums these in a chain of $n-1$ additions. If b is a constant, one can omit all additions where $a \cdot b_i$ with $b_i = 0$ is added. To have a gain also for numbers with many ones in their binary representation, one uses the identity

$$\sum_{i=0}^{n-1} b_i \cdot 2^i = 2^n - 1 - \sum_{i=0}^{n-1} \bar{b_i} \cdot 2^i \ . \tag{1}$$

If the binary representation of b has more than $n/2+1$ ones, then one forms partial products according to the right side of identity (1). One needs 2 subtractions and at most $n/2-3$ additions, because the bitwise inverse of b has at most $n/2-2$ ones. If the binary representation of b has at most $n/2+1$ ones, then one forms partial products according to the left side of identity (1). One needs at most $n/2$ additions. In any case the depth of the array multiplier is approximately halved, which means an enormous amount of acceleration. There are even better optimizations, e.g. Bernstein's algorithm [Bernstein (86)], which are however more difficult to analyze.

In an application note for Xilinx field-programmable gate arrays (FPGAs) [Chapman (96)], a circuit to multiply a constant and a 4-bit variable is described. The circuit has constant depth, it uses a number of 16-bit RAM-based lookup-tables which Xilinx FPGAs generally use to implement their functionality. These are addressed with the 4-bit variable, each stores a particular bit of the result depending on the value of the variable. When multiplying with an n-bit variable, only $n/4$ partial products have to be added, giving an additional reduction by a factor of 2 compared to the previous scheme.

A Wallace-tree multiplier sums the partial products in a balanced binary adder tree. If the number of partial products to be added is reduced to about one half by using the transformations described above, then one saves one stage of adders. In the gate model, thus only a small amount of acceleration can be expected, because all stages except the last use a redundant number representation. This means that the depth of the last adder stage is as large as the sum of the depths of all previous stages. E.g. for a 16-bit multiplication there are at most 16 partial products which are summed in a tree of 4 stages, where the first

three stages contribute about one sixth each of the multiplier's depth and the last stage contributes the remaining half. Hence, omitting the first stage reduces the depth by one sixth. In the VLSI model however, where wire lengths and wire delays are also counted, a larger amount of acceleration is to be expected, see [Paul and Seidel (98)].

3.3 AES

The purpose of the Advanced Encryption Standard program (AES) announced by the US National Institute of Standards and Technology (NIST) [NIST (00)] is to select a successor to DES. Security as well as efficiency on a variety of platforms (high- and low-end microprocessors, smartcards, dedicated hardware) plays an important role in the selection process. It is expected that AES will replace DES as the standard encryption algorithm. Apart from being a 128-bit block cipher, requirements for AES include key lengths of 128, 192 and 256 bits and the absence of weak keys. The fifteen official AES candidates were selected from a pool of public submissions and announced last year. After a first round of public comments, the five finalist ciphers were selected: MARS, RC6, Rijndael, Serpent and Twofish. These ciphers were thoroughly evaluated during the second round of public comments. This round has been closed recently, but up to now the final AES cipher has not been announced. We investigate the five finalist ciphers with respect to our concept in the following paragraphs.

Key scheduling is used by all investigated algorithms in order to generate the required round keys from a single secret key. As the generation of round keys depends only on the secret key, the resulting round keys can be pre-computed for a specific key and stored in a table. The complexity of key scheduling and therefore the potential savings differ significantly between algorithms. For example, key scheduling in hardware takes between 0 ns (Rijndael) and 10000 ns (MARS) in the case of encryption. The corresponding values for decryption range from 60 ns (Twofish) to 30.000 ns (MARS). These values are based on performance estimations performed on VDHL implementations of all ciphers [Weeks et al. (00)].

MARS has a structure different from the other ciphers: MARS uses a mixed structure of a "cryptographic core" embedded in two "wrapper layers". The first wrapper layer consists of a key addition (128 bit) followed by 8 rounds of unkeyed, i.e. without any key input, mixing using a Feistel network. The cryptographic core consists of 16 rounds of keyed transformations using another Feistel network. Each round uses two keyed expansion functions, that utilize one key addition (32 bit) and one key multiplication (32 bit). The second wrapper layer consists of 8 rounds of unkeyed mixing transformations using a Feistel network followed by key subtraction (128 bit). Overall, eight of the total twenty-eight 32-bit adders in the wrapper layers as well as thirty-two of the total ninety-six 32-bit adders and all thirty-two 32-bit multipliers in the cryptographic core can be optimized. Similar savings are possible for decryption. MARS therefore has significant potential savings with regards to our concept.

RC6 is a family of encryption algorithms that differ by word length, key length and number of rounds. The AES submission is targeted at 32-bit word lengths, 20 rounds and 128, 192 or 256 bit keys. RC6 uses two initial key additions followed by two key additions per round as well as two final key additions, which yields a total of 44 additions available for optimization. In the case of

decryption, addition is replaced by subtraction, therefore the argument holds for both operations.

Rijndael is an iterated block cipher using variable block sizes and key sizes of 128, 192 and 256 bit. After an initial key combination using exclusive OR, between 10 and 14 rounds are performed. The exact number of rounds depends on block and key sizes. Each round contains four steps: a non-linear byte substitution, two linear mixing operations and a key combination using exclusive OR. The final round of Rijndael contains only one of the linear mixing operations. With respect to our concept, all, i.e. up to 15, key combinations can be optimized by omitting/replacing the individual gates. Similar savings are possible for decryption.

Serpent is a 32-round substitute-permutate (SP) network cipher that operates on four 32-bit values in parallel. The cipher contains an initial permutation followed by 32 rounds and another permutation at the end. Each round consists of a key combination using exclusive OR, a non-linear byte substitution and a linear transformation. The last round replaces the linear transformation with an additional key combination using exclusive OR. Since both permutations do not depend on the key, all 33 exclusive OR key combinations can be optimized by omitting/replacing the individual gates. Similar savings are possible for decryption.

Twofish uses a 16-round Feistel network that operates on four 32-bit words in parallel. Key operations include input and output whitening (eight 32-bit exclusive-ors) as well as two 32-bit additions per round. Therefore all eight 32-bit exclusive-ors and thirty-two of the total sixty-four 32-bit adders can be customized. Similar savings are possible for decryption.

The AES finalists presented above cover a wide spectrum of potential savings: Apart from key setup, the benefits to encryption/decryption speed range from low (RC6, Rijndael, Serpent) to significant (MARS, Twofish).

3.4 RSA

The RSA algorithm is the most prominent representative of public key algorithms. Each receiver has a public key (n, e) and a private key d. Each message x is encrypted by computing $x^e \bmod n$. To reduce the time to encrypt a message, many tools fix e to 3 or $2^{16} + 1$. An encrypted message y is decrypted by computing $y^d \bmod n$. Both encryption and decryption are dominated by exponentiation and modulo computation, especially because the size of n is at least 512 bits for security reasons.

Exponentiation can be accelerated by applying addition or division chains [Walter (98)]. The exponent is represented by a sum of terms in such a way that exponentiation needs as few multiplications as possible. Computation of optimal addition or division chains is NP-hard. Therefore, simple heuristics are used that produce suboptimal chains. An optimization consists of finding an optimal or close to optimal chain for that key. Thus, we can assume that a description of a generic RSA encryption/decryption circuit is held locally for each user of the system and that the key directory only stores the description of a chain for each user, or one chain if e is fixed. For decryption, each user locally stores a chain corresponding to d. As these chains are computed once and offline, much better heuristics can be used.

A further improvement is possible by replacing fast exponentiation and repeated modulo computation on intermediate results by applying a Montgomery-Reduction first, see [Menzes et al. (97)].

4 Summary and Outlook

In this article we sketch the (as far as we know) new idea to use key-optimized algorithms for high-speed encryption and decryption of large sets of data such as real-time video streams, which can be processed on low-priced and general purpose programmable hardware as an alternative to the usage of special purpose hardware.

Future investigations will consist in the experimental verification of the expected efficiency gain by usage of customized encryption algorithms together with Xilinx FPGAs compared to traditional encryption processes. For a widespread applicability of our method, it will also be necessary to use a standardized, compact and device independent description for hardware, that additionally allows for the download into a programmable hardware device without much processing. While the first three properties are fulfilled by VHDL, the last is not.

A potential application of customized encryption and decryption algorithms is their usage in combination with smartcards. At present smartcards are frequently completely configured by the provider before they are issued to the users, i.e. the algorithms used and in most real-world applications even the secret keys are stored on the smartcard and cannot be changed by the users. Due to the relatively small performance of smartcard microprocessors and the limited storage usually only one encryption algorithm is stored in the smartcard ROM. Current developments such as the Java-smartcards and for example the Mondex-System are examples for the trend to implement interpreter programs in smartcard ROMs and to load application-specific programs in this interpreter language onto the card as soon as a new application is needed. One can imagine applications, in which it is favourable that the smartcard owners may choose their preferred encryption algorithm and secret keys by themselves. Then the usage of customized decryption or signature algorithms might be a valuable method to cope with performance problems. Certainly, one has to take attention to the storage limitations of the smart card, i.e. descriptions of customized algorithm have to be of small size. Our future work will investigate the tradeoff between efficiency improvements and higher storage costs as well as reasonable practical applications for customized encryption algorithms on smartcards.

References

[Bernstein (86)] Bernstein, R.: "Multiplication by Integer Constants"; Software — Practice and Experience, 16, 5 (1986), 641-652.

[Chapman (96)] Chapman, K.; "Constant Coefficient Multipliers for the XC4000E"; Xilinx Application Note (XAPP 054), Xilinx Inc. (1996)

[EFF (98)] Electronic Frontier Foundation (EFF): "Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design"; O'Reilly & Associates, Sebastopol (1998)

[Hauser and Wawrzynek (97)] Hauser, J. R., Wawrzynek, J.: "Garp: a MIPS processor with a reconfigurable coprocessor"; Proc. IEEE Symp. on FPGAs for Custom Computing Machines, IEEE Press, Los Alamitos (1997), 12-21.

[Menzes et al. (97)] Menezes, A. J., van Oorshot, P. C., Vanstone, S. A.: "Handbook of Applied Cryptography"; CRC Press, Boca Raton (1997)

[NIST (00)] National Institute of Standards and Technology (NIST): "Advanced Encryption Standard (AES) Development Effort"; http://csrc.nist.gov/encryption/aes/ (2000)

[Paul and Seidel (98)] Paul, W. J., Seidel, P.-M.; "On the complexity of booth recoding"; Proc. 3rd Conf. on Real Numbers and Computers (RNC3), 199-218.

[Schneier (95)] Schneier, B.; "Applied Cryptography, 2nd edition"; John Wiley & Sons, New York (1995)

[Schneier et al. (98)] Schneier, B., et al.; "Twofish: A 128-Bit Block Cipher"; Manuscript (1998), http://www.counterpane.com/twofish-paper.html

[Schneier et al. (99)] Schneier, B., et al.; "Performance Comparison of the AES Submissions"; Manuscript (1999), http://www.counterpane.com/aes-performance.html

[Walter (98)] Walter, C. D.; "Exponentiation Using Division Chains"; IEEE Transactions on Computers, 47, 7 (1998), 757-765.

[Weeks et al. (00)] Weeks, B., Bean, M., Rozylowicz, T., Ficke, C.; "Hardware Performance Simulations of Round 2 Advanced Encryption Standard Algorithms"; Manuscript (2000), http://csrc.nist.gov/encryption/aes/round2/r2anlsys.htm