Protection of software algorithms executed on secure modules

ELSEVIER

Future Generation Computer Systems 13 (1997) 55–63

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

Protection of software algorithms executed on secure modules[†]

H.D.L. Hollmann[a,*] J.P.M.G. Linnartz[a,1], J.H.van Lint[b], C.P.M.J. Baggen[a,2], L.M.G. Tolhuizen[a,3]

[a] *Philips Research Laboratories. Prof. Holstlaan 4, WY8, 5656 AA Eindhoven, Netherlands*
[b] *Eindhoven University of Technology, Eindhoven, Netherlands*

Abstract

Loop structures in software code may reveal essential information about implemented algorithms and their parameters, even if the observer has no knowledge about which instructions are executed. Regular patterns can for instance be observed in power consumption, instruction fetches in external memory, or radiated EM energy. This paper addresses the use of dummy operations to obscure the details of the algorithm executed by the processor. We show that for a particular class of dummy insertion strategies, a Viterbi decoder can fairly reliably distinguish dummy fetches from real instruction fetches.

In the second part of this paper, we study strategies to choose dummy fetches from a more general model. For certain situations, the optimum protection strategy appears to be deterministic (as opposed to random). Moreover, we show that in such a case, it is fundamentally not possible to enhance the security of the implementation by keeping the strategy for generating dummy fetches secret to the attacker.

Keywords: Software protection; Secure processor; Viterbi decoder; Dummy instructions

1. Introduction

Most investigations into the strength of cryptographic algorithms assume the encryption or decryption algorithm to behave as a black box with three well-defined ports to the external world: one input and one output for the user data and an input for the cryptographic key. A classical situation is the chosen plaintext attack in which an attacker can send input data of his own choice to the circuit and can observe the encrypted output (ciphertext). This scenario is often used to evaluate attacks on smart cards with the objective to find the hidden keys buried in the silicon circuit on the card.

Practical implementations, however, may be less ideal than such a black box model, because significant amounts of information about the key may leak through necessary actions of the processor during execution of the cryptographic algorithm. Execution times, currents drawn through power supply pins, radiated electromagnetic energy and accesses to an external memory for fetching instructions can all be exploited by an attacker.

[†] This paper was presented at the 2nd CARDIS Conference, September 1996, Amsterdam.
[*] Corresponding author. E-mail: hollmann @natlab. research. philips.com.
[1] E-mail: linnartz @natlab. research. philips.com.
[2] E-mail: baggen @natlab. research. philips.com.
[3] E-mail: tolhuizn @natlab. research. philips.com.

Recently, it has been realized (e.g. [1]) and demonstrated that observed computation times used by an RSA public-key crypto system [3] might be exploited to find the private key used by the processor. In RSA, a plain text message m is raised to some power e, where e is a cryptographic key. Typically, the operation $c = m^e \bmod N$ is implemented on a binary machine through successive squaring of an intermediate result $\hat{m}$. The algorithm recursively reads all key bits. For a bit in the key e that is set ("1"), the processor executes at least one multiplication instruction more than for a "0" bit. If the attacker is able to determine when such instructions are executed, the secret key e can be found.

This example appears to be a special case of a larger class of possible attacks to reconstruct embedded algorithms in electronic circuits. Therefore, in this paper we widen our scope, including not only smart cards but also security modules that consist of several chips, connected with buses that can be probed.

Since conventional processors provide almost no possibilities to keep the algorithm or its parameters confidential, smart cards and "secure processors" [4–6] are commonly used to allow execution of algorithms without revealing critical details to external observers. The central processor unit (CPU) of a secure processor accesses an external memory to fetch encrypted instructions. Decryption occurs on the CPU chip. This scenario has the advantage that the manufacturer or its service organization can easily replace the (cheap) ROM memory to update the features of the system, without having to replace other, more expensive modules.

Secure processors often play exactly the same role as smart cards, i.e., providing a secure shell in an insecure system, except that secure processors are not detachable but hardwired into the system. Secure processors are for instance used in the ignition, steering and braking control of automobiles. The automotive industry desires to keep their proprietary algorithms confidential to their competition, and does not want dangerous situations to evolve if hobbyists "tune up" the performance of their vehicle by tampering with the algorithms.

Even if the attacker of the secure processor is not able to break the encryption/decryption algorithm, he may get substantial information about the algorithm performed and its parameters from the loop structure of the algorithm. These can be obtained through periodic patterns in the addresses of instructions fetched from memory. Particularly if the above exponentiation algorithm is executed, the trace of addresses reveals which bits in the key have been set to "1". Attempts to obscure the loop structure by a secret permutation on the location of instructions on the memory chip slows down the memory access, and by itself it cannot obscure repetitive patterns in addresses.

Information leakage can be avoided if one allows that the processor always reads all instructions in the external memory sequentially, but only uses the instructions needed. Such a solution severely limits the performance of the implementation, so mostly this is not realistic or desirable.

This paper reports new results on minimizing the amount of information leaked in such scenarios. The organization is as follows: Section 2 models the process of fetching instructions from memory as a Markov process. This suggests to use a Viterbi decoder to trace the most likely path, which is worked out in Section 3. A more generic case is presented in Section 4 and evaluated in Sections 5 and 6. Section 5 derives a lower bound for the probability that the attacker can guess the program sequence correctly. Section 6 shows that a particular strategy can be chosen to ensure optimum security. In this case, the probability that the attacker can guess the correct program sequence is minimum. As an example, Section 7 addresses the case of an n round sequential execution of an algorithm having $n-k$ rounds of relatively short duration and k rounds of long duration.

2. Dummy reads

As illustrated in Fig. 1, a secure processor consists of a single chip with a microprocessor and a decryption circuit for incoming instructions. The decryption key depends on the address of the instruction. Typically a (secret) random number is generated, using the address as a seed.

It has been proposed to insert dummy fetches (only) when the processor is busy executing previously fetched instructions or instructions

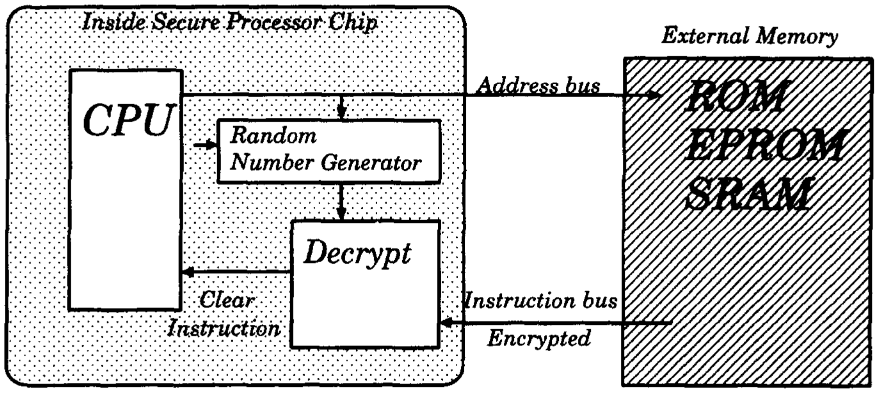


Fig. 1. Basic elements of secure micro processor. The CPU and decryption are implemented on the same chip, using a silicon technology that is difficult to reverse engineer.

that are already in the processor core (or its cache memory).

In order to investigate such dummy reads, it appears reasonable to model the CPU as a finite state machine, which asks for the next instruction depending on its current state. In Sections 2 and 3 we will make the simplification that the sequence $X^v = X_0, X_1, \ldots, X_v$ of addresses of instructions is a time-homogeneous Markov process. That is, the probability that $X_{v+1} = j$ depends only on X_v, but is further independent of the (discrete) time v. The unit of time can for instance be the length of a clock cycle on the bus connecting the CPU chip with its memory. In Section 4 we broaden this model.

Now let $p_{i,j}$ denote the transition probability of $X_{v+1} = j$ given $X_v = i$. Whenever $i = j$, the processor is busy executing instruction X_v during cycles v and $v+1$, and X_{v+1} may on the external address bus be replaced by an arbitrary dummy read Y_{v+1}. Otherwise, if $i \neq j$ then $Y_{v+1} = X_{v+1}$. We use $Y^v = Y_0, Y_1, \ldots, Y_v$ to denote the sequence of (dummy and real) addresses that can be observed on the address bus.

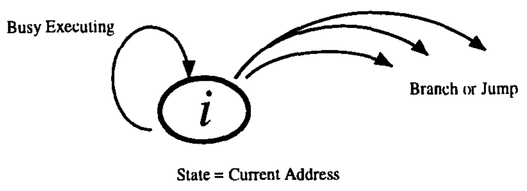


Fig. 2. Markovian model for instruction addresses in software execution.

Fig. 2 describes a postulated model for instruction addresses X^v. The parameters have been inspired by rules of thumb used in processor design, such as "about one in five instructions is a jump", i.e., for all $i, p_{i,i+1} \approx 4\sum_j p_{i,j}$, with $j \neq i, i+1$. and "75% of time the processor is busy executing previously loaded instructions" ($p_{i,i} \approx 0.75$).

3. Estimation of hidden Markov process

An attcker must estimate the original sequence X^v from the observed Y^v. Under the condition that the $Y^v|X^v$ is a sequence of independent random variables, that is, the choice of the dummy read Y_v depends only on X_v but not on X^{v-1}, the problem reduces to that of estimating a hidden Markov process. In particular, the most likely path can be found using

$$\hat{X}^v = \arg\max_{X^v} \operatorname{Prob}(X^v \mid Y^v),$$

that is, it finds the argument X^v that has maximum a posteriori probability, given the observed Y^v. Using Bayes' rule, this expression is rewritten as

$$\hat{X}^v = \arg\max_{X^v} \frac{\operatorname{Prob}(X^v \mid Y^v)\operatorname{Prob}(X^v)}{\operatorname{Prob}(Y^v)},$$

where we may omit $\operatorname{Prob}(Y^v)$ because it does not influence the choice of X^v that maximizes the expression. Moreover, using the Markovian property of the instruction sequence X^v and the independence of $Y_v|X_v,X_{v-1}$ on X^{v-2}, and after taking a logarithm, we arrive at

$$\hat{X}^v = \arg\max_{X^v} \sum_{w=0}^{v} [\log \mathrm{Prob}(Y_w|X_w,X_{w-1}) + \log \mathrm{Prob}(X_w|X_{w-1})]$$

where we define $Prob(X_0|X_{-1})$ as the initial (a priori) probability $Prob(X_0)$. The above expression has been the basis to the architecture of the Viterbi decoder, which gives an efficient way to find the sequence $\hat{X}^v$ that has maximum probability using dynamic programming. The complexity of the Viterbi decoder is of the order of the number of states visted by the underlying Markov Process. In our particular application, the decoder continuously administers for every possible address i a matrix related to the probability that the algorithm is at that address ($\mathrm{Prob}(X_v = i)$). For a detailed review of this decoder we refer to [7] or one of the many text books on communication theory.

In Fig. 3, we use transition probabilities from our postulated model on the a priori behaviour $p_{i,j}$ of software algorithms, and we use $\mathrm{Prob}(Y_v|X_v, X_{v-1})$ from the assumption that dummy reads are randomly chosen whenever $X_v = X_{v-1}$ with equal (uniform) probability for all memory addresses.

In Fig. 3, the program proceeds with the next instruction (or with the next cache line) with probability $p_{i,i+1} = 0.2$. With probability 0.05, a jump to an instruction with a uniformly random location occurs. The probability that the CPU does not fetch a new instruction is 0.75, which is equal to the fraction of instructions fetched on the external bus that are dummy reads. We take $X_0 = 0$ with probability 1.

In this example we see that the decoder only makes errors between clock cycles 10 and 20. In the rest of the trace, the estimated path (·–) coincides with the path that the CPU actually followed through the instructions.

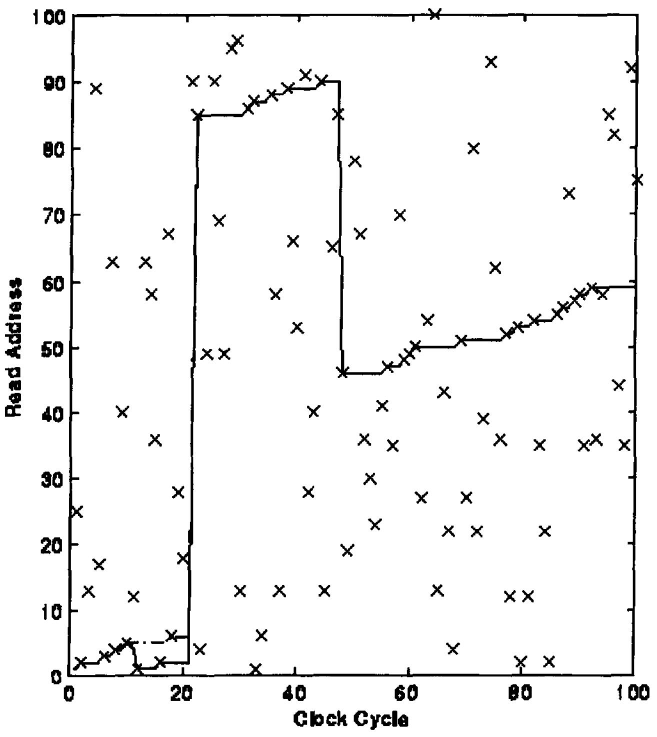


Fig. 3. Instruction address as a function of time. Sample path of the address (solid line) seen by CPU, instruction fetches on bus, including both real and dummy reads (crosses), and the most likely path as estimated by a Viterbi Decoder (·–·–).

Speaking in terms of hidden Markov processes, the software developer wishes to minimize the mutual information $I(X^v; Y^v)$. Explicit computation of the conditional entropy $H(X^v|Y^v)$ or $I(X^v;Y^v) = H(Y^v) - H(Y^v|X^v)$ is known to be tedious for hidden Markov processes. Moreover, the designer may select more intricate strategies for choosing Y^v than is covered by the hidden Markov model. In such cases the Viterbi algorithm may not be applicable. In the next section, we broaden the class of dummy generation strategies. It turns out that we can quantify the uncertainty in X^v given the observation Y^v for some special cases.

4. Generic model

In what follows, we will refer to X^v as a *program sequence* of length v. To encrypt such a sequence, entries at locations n for which $X_n = X_{n-1}$ may be replaced by another, according to some method, the *encryption strategy*, that is specified beforehand. We will call a location n with $X_n = X_{n-1}$ a "star", as we may replace the entry X_n by a "wild-card" $*$. (By abuse of notation, we will denote both the original program sequence and the sequence obtained after these replacements by X^v.) Regardless of whether this is desirable from a security point of view, at this point we will allow this encryption method to be *non-deterministic*, that is, the same sequence may be encrypted differently, at different moments or by different realizations of the processor. (Later, we will see examples where the best encryption strategy turns out to be deterministic after all.) We can thus describe an encryption strategy by specifying, for each program sequence $X^v = s$ and each possible encryption $Y^v = t$ of s, the probability that s will be encrypted as t. The task of the interceptor is to obtain information about the original program sequence from the encrypted version.

We can formalize the above idea as follows. Let S be the collection of all possible program ("source") sequences, and let T denote the collection of all possible images (encrypted sequences). Write $|S|$ to denote the cardinality of S. Note that this model is not just limited to instruction fetches by a secure processor. T may for instance denote the observed power consumption pattern of a smart card, while S is the set of possible RSA private keys.

An *encryption strategy* is described by an $|S| \times |T|$-matrix Q, where $Q_{s,t} = p(t|s)$ is the probability that a given program sequence s will be encrypted as the sequence t. Obviously, the attacker cannot do better than following a Maximum A Posteriori (MAP) probability decoding method: given an encryption t, the original program sequence is estimated as

$$\hat{s} = \arg\max_{s \in S} p(s|t).$$

(Note that, in general, the attacker would have to know the encryption strategy Q in order to do so.) In this case, the optimal (MAP) probability p of correct decoding, averaged over all t, is

$$p = \sum_{t \in T} p(t) \max_{s \in S} p(s|t) = \sum_{t \in T} \max_{s \in S} p(t|s)\, p(s),$$

where $p(s)$ denotes the probability that the program sequence to be encrypted equals s. Our aim is to choose Q such that p is as small as possible, since the attacker on an average has the uncertainty of at least $1/p$ program sequences that could have resulted in the observed $Y^v = t$. In the special case of equiprobable program sequences, i.e., if $p(s)$ is independent of s, we have $p(s) = 1/|S|$ and the detection principle reduces to maximum likelihood detection, with

$$p = |S|^{-1} \sum_{t \in T} \max_{s \in S} p(t|s).$$

Note that this p is an *average* probability, that is, some sequences may be easier to decrypt than others.

5. A method to obtain a lower bound

We will now show that p cannot be made arbitrarily small by choosing a suitable Q. To this end, let $A \subset S$ be a collection of source sequences such that any two distinct words in A will always be encrypted into distinct codewords. (Note that this is precisely the case when we can find for each two distinct sequences in A a (non-star) coordinate where these sequences have different non-star symbols.)

Now a possible approach by the attacker is to decrypt when possible an observed sequence into the corresponding program sequence a from A. (Note that by our assumption on A there can be at most one such sequence.) Using this approach, the probability of correct decoding at least equals the probability that the program sequence is contained in A. So we have proved that

$$p \geq \sum_{a \in A} p(a).$$

In the case of equiprobable program sequences, this bound reduces to

$$p \geq |A|/|S|.$$

6. A case of equality

We will now discuss a situation where the lower bound in the previous section holds with equality. For the remainder of this section, we will assume that the program sequences are equiprobable. Let A be a collection as in the previous section. To each $a \in A$ we will assign a *fixed* encryption $t_a \in T$. Now suppose that this assignment can be done in such a way that the collection $B = \{t_a | a \in A\}$ has the property that *each* program sequence can be encrypted into some member of B. Then, as a consequence of these assumptions we can assign to each program sequence $s \in S$ an encryption $f(s)$ in B, with $f(a) = t_a$, $a \in A$. Note that, due to our assumptions on A, the sequences $t_a, a \in A$, are all distinct, that is, $|B| = |A|$. We will consider the map f as describing a *deterministic* encryption rule.

The corresponding strategy matrix Q is given by

$$p(t|s) = \begin{cases} 1 & \text{if } t = f(s), \\ 0 & \text{otherwise.} \end{cases}$$

We now claim that under these assumptions, we have that

$$p = |A|/|S|,$$

with optimal encryption strategy described by Q (or by f). Indeed, we have that

$$p = \sum_{t \in T} \max_{s \in S} p(t|s)/|S|$$
$$= \sum_{b \in B} 1/|S| = |B|/|S| = |A|/|S|.$$

We conclude that the optimal encoding strategy can be deterministic. The result also shows that choosing a statistical strategy does not always make the task of the attacker more difficult.

7. Example

As a special case of the theory developed in the previous sections, consider the sequential execution of n rounds of an algorithm out of which some, say k steps require a longer execution time. A program is represented by a sequence X^v of length $v = n + k$ obtained from the sequence

$$123\ldots n$$

by the insertion of precisely k stars. Moreover, the first and last symbols in the sequence are non-stars, and the sequence does not contain two consecutive stars.

Let S be the collection of all source sequences with n symbols and k stars, and let T denote the collection of all possible images of source sequences in S. Write $|S|$ or $|S|(n,k)$ to denote the cardinality of $S(n,k)$. Note that $|S|(n,k) = \binom{n-1}{k}$.

For a word s in S, let $*(s)$ denote the collection of positions where the source sequence s has a star. For example, if

$$s = 12{*}3{*}45,$$

then

$$*(s) = \{3, 5\}.$$

To simplify the following arguments, we imagine that the collection of positions $I = \{1, 2, \ldots, n + k\}$ is partitioned into pairs

$$I_1 = \{1,2\}, \quad I_2 = \{3,4\}, \ldots,$$

and possibly (if $n + k$ is odd) the singleton set $I_0 = n + k$. (Note that, if $n + k$ is odd, then all words s in S will have the symbol n in position $n + k$, hence all stars in words of S occur within the sets $I_j, j > 0$.) We take the set A as the collection of all a in S for which $*(A)$ consists of even numbers only. Note that we have

$$|A| = \binom{[(n+k-1)/2]}{k}.$$

It can be reasoned that any two members of this A will always produce distinct codewords. The encoding map f is described in terms of the coordinate pairs I_j as follows. For all sequences s in S, replace a star in position $i \in I_j$, say, by the symbol on the other position of I_j. The resulting t sequence could also have been generated from a sequence with stars only in even positions. It is now evident that

$$f(s) = f(a(s))$$

holds for every word s in S, where $a(s)$ denotes the special sequence in A that has the same image as s. So we have proved that the probability of correct decoding

$$p = \binom{[(n+k-1)/2]}{k} \bigg/ \binom{n-1}{k} = \frac{[(n+k-1)/2]!(n-k-1)!}{[(n-k-1)/2]!(n-1)!}$$

holds for each n and k, with the optimal deterministic encoding strategy given by the map f above.

7.1 Asymptotic behavior

To study the asymptotic behavior for large programs ($n \to \infty$), we define the ratio $\alpha = k/n$. Appendix A shows that

$$p \to 2^{n(1-\alpha-h(1/2+\alpha/2))}.$$

The optimum ratio α which asymptotically minimizes the value of the probability of correct decoding p with a given n and variable $n+k$ appears to be $\alpha = k/n = \frac{3}{5}$. The optimum density of stars in very long programs equals $\frac{3}{8}$. This leads to

$$\lim_{n\to\infty} \frac{1}{n} \log p = \frac{4}{5} h\left(\frac{3}{4}\right) - h\left(\frac{3}{5}\right) \approx -0.322,$$

where $h(q)$ is the binary entropy function. Hence, the probability of correct decoding tends to $p = 2^{-0.322n}$.

We interpret this as "every three rounds reveal about one bit of information". If the model applies to particular observations made on a smart card running a computation of 512 bit RSA, the attacker would have an uncertainty of about 165 bits in estimating the key. At this moment, we are unaware of methods that exploit such information in breaking this cryptographic algorithm.

8. Concluding discussion

We investigated leakage of information through observations made on the execution of algorithms. The results apply to smart cards, secure processors and other detachable or fixed security modules.

The address bus between a secure processor and its memory is a weak point in the security of the system. Substantial information can be obtained from observing the loop structure of the program executed by the secure processor.

The insertion of dummy instructions helps to obscure the exact structure of the program. However, methods exist to separate dummy reads from real instruction reads. We tested the use of a Viterbi decoder on simulated address traces and concluded that dummy reads may not simply be chosen uniformly randomly.

A frame work for the analysis of dummy reads has been proposed. We used this framework to find results for a special case of software code. For our considered situation, optimum strategies are deterministic. In such cases, revealing the attacker information about the strategy for generating dummy reads does not compromise the security of the secure processor.

Although our results are only a preliminary step towards a full understanding of these issues, the results are believed to be relevant to evaluate the strength of secure processors, to develop stronger or more efficient strategies for dummy fetches and to better understand the potential of recently introduced timing attacks (in which cryptananalysts exploit measurements of instruction execution times).

Acknowledgements

The authors wish to thank P. Lenoir for fruitful discussions.

Appendix A: Asymptotic star density

To study the asymptotic behavior for large programs ($n \to \infty$), we use the binary entropy function, defined as

$$h(q) = -[q \log q + (1-q)\log(1-q)]. \qquad \text{(A.1)}$$

Its derivative is

$$h'(q) = \log \frac{1-q}{q}, \tag{A.2}$$

for any base of the logarithm. In this paper, we assume all logarithms and entropy functions to the base 2.

Using Stirling's approximation for factorials

$$\sqrt{2\pi n} n^n e^{-n} e^{-1/(12n+1)} < n! < \sqrt{2\pi n} n^n e^{-n} e^{-1/12n}, \tag{A.3}$$

we obtain for binomial coefficients

$$\binom{n}{k} = 2^{n[h(k/n) + O(\log n/n)]}. \tag{A.4}$$

Inserting this in the expression for p, we obtain

$$\lim_{n\to\infty} \frac{1}{n} \log p = \frac{1+\alpha}{2} h\left(\frac{2\alpha}{1+\alpha}\right) - h(\alpha) \tag{A.5}$$

$$= 1 - \alpha - h(\tfrac{1}{2} + \tfrac{1}{2}\alpha). \tag{A.6}$$

We are interested in the value of k which asymptotically minimizes the value of probability of correct decoding p. In order to find the optimum (for a given n and variable $n+k$), we set the derivative with respect to α equal to zero:

$$-1 = \frac{1}{2} h'\left(\frac{1}{2} + \frac{\alpha}{2}\right) = \frac{1}{2} \log\left(\frac{1-\alpha}{1+\alpha}\right).$$

We find that the optimum α equals $\alpha = \frac{3}{5}$. Note that the optimum density of stars in very long programs equals $\frac{3}{8}$, since the total length of a string including the stars equals $(1+\alpha)n$. Substituting this result in (A6), we obtain

$$\lim_{n\to\infty} \frac{1}{n} \log p = \frac{4}{5} h\left(\frac{3}{4}\right) - h\left(\frac{3}{5}\right). \tag{A.7}$$

References

[1] P.C. Kocher, Cryptanalysis of Diffie-Hellman, RSA, DSS and other systems using timing attacks, *Proc. Crypto96*, (Santa Barbara, August 1996).

[2] B. Schneier, *Applied Cryptography*, (Wiley, New York, 1996).

[3] R.L. Rivest, A. Shamir and L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, *Comm ACM*, 21(2) (1978) 120–126.

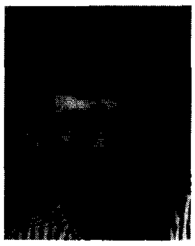
[4] R.M. Best, Preventing software piracy with cryptomicroprocessors, *Proc. Compcon* IEEE-CS Press, Los Alamitos, CA, 1980) 466–469.

[5] R.M. Best, US Patent 4168396, Microprocessor for executing enciphered programs, 18 September 1979.

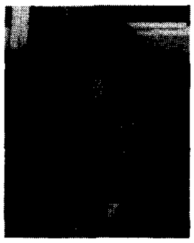
[6] US Patent 5386469, Firmware encryption for microprocessor/microcomputer, 31 January 1995.

[7] G.D. Forney, Jr., The Viterbi algorithm, *Proc IEEE*, 61, (1973) 268–278.

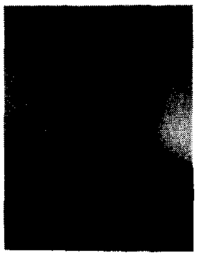
Henk Hollmann was born in Utrecht, Netherlands, on 10 March 1954. He received his masters degree (cum laude) in Mathematics from Eindhoven University of Technology, with a thesis on association schemes. In 1982 he joined CNET, Issy-les-Moulineaux, France, where he worked mainly on Number Theoritic Transforms. Since 1985 he is with Philips Research Laboratories, Eindhoven, Netherlands. In 1996, he recieved a Ph.D. degree from Eindhoven University of Technology with the thesis "Modulation codes". His research interests include discrete mathematics/combinatorics, information, cryptography, and digital signal processing.



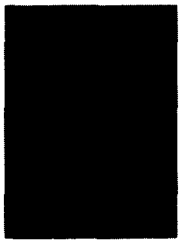
Jean-Paul M.G. Linnartz received his Ir. (M.Sc. E.E.) degree in Electrical Engineering (Cum Laude) from Eindhoven Univeristy of Technology, Netherlands, in 1986. During 1987–1988, he worked with the Physics and Electronics Laboratory (F.E.L-T.N.O., The Hague) of Netherlands Organization for Applied Scientific Research on frequency planning and UHF propagation. From 1988–1991, he was Assistant Professor at Delft University of Technology, where he received his Ph.D. (Cum Laude) on multi-user mobile radio nets in December 1991. Since January 1992, he has been an Assistant Professor in the Department E.E.C.S. at the University of California at Berkeley, USA. In 1994, he returned to Delft University of Technology in Netherlands, as an Associate Professor. Currently he is with Philips Natuurkundig Laboratorium, Eindhoven, Netherlands. His main research interests are conditional access and information security, electronic watermarks, (wireless) multi-media communications, communication over fading channels and multicarrier CDMA. In 1993, he published the book "Narrowband Land-Mobile Radio Networks". He is Editor-in-Chief of "Wireless Communications, The Interactive Multimedia CD ROM" His URL is "http://www.eecs.berkeley.edu/~linnartz".



Jacobus H. van Lint was born in Bandung, Indonesia, on 1 September 1932. He received the M.Sc. in Mathematics and Physics and the Ph.D. degree in Mathematics from the University of Utrecht in 1955 and 1957, respectively. In 1959 he became the Professor of Mathematics at Eindhoven University of Technology where he ended his career as rector magnificus in the period February 1991–September 1996. In 1966, 1971, and 1977 he was a temporary Member of Technical Staff at Bell Laboratories, Murray Hill, N.J. During 1970–1971 he was Morgan Ward Visiting Professor at the California Institute of Technology, Pasadena, CA, where he was a Fairchild Distinguished Scholar during 1982–1983 and Visiting Professor in 1988–1989. He has been a consultant at Philips Research Laboratories since 1985. His research interests include combinatorics, coding theory, and number theory. He is the author of several books, including *Introduction to Coding Theory and A Course in Combinatorics* (with R.M. Wilson). He is a member of the Royal Netherlands Academy of Arts and Sciences. He received honorary doctorates from Bergen University (Norway) and the Technical University of Bucharest (Romania).



Stan Baggen was born in Grubbenvorst, Netherlands, on 22 March 1953. He received the Ingenieurs (Ir.) degree from the Eindhoven University of Technology in 1979. The Ir. thesis research was conducted at the Institute of Perception Research (IPO), Eindhoven, in the area of cognition. In 1979, he joined the Philips Research Laboratories in Eindhoven, where he is currently a senior scientist. His main research interests are in the area of algebraic coding, communication and information theory, and applications thereof to storage and transmission of digital information. He spent seven years in the optical recording group, where he designed the error correction system of the first generation 12″ optical recorders. He was involved in the design of error correction decoders for CD players and in standardization discussions of CD-ROM third layer error correction. In 1986, he moved to the digital signal processing group, where he became responsible for research on channel coding. He spent the academic year 1989–1990 with Prof. Jack Wolf at the Center for Magnetic Recording Research (CMRR), at UCSD, San Diego, where he started working towards a Ph.D. He received his Ph.D. in Electrical Engineering (Communication Theory and Systems) from the UCSD in 1993. Recently, he has been involved in the design of channel coding systems for Digital Video Broadcast (DVB), and for the Digital Versatile Disc (DVD) system.



Ludo Tolhuizen was born on 23 June 1961, in Roosendaal, Netherlands. In 1986, he obtained the M.Sc. (Ir.) degree (with honours) from Eindhoven University of Technology with a thesis on error correcting codes. After that, he joined Philips Research Laboratories in Eindhoven where he has been working on theoretical and practical issues in error correcting codes. In 1996, he received the Ph.D. degree from Eindhoven University of Technology with the thesis "Cooperating error-correcting codes and their decoding". The research interest of Ludo Tolhuizen include error-correcting codes, information theory, combinatorics, cryptography and digital signal processing. He is a member of the IEEE Information Theory Society.