A speed-up theorem for cryptosystems. (2000,unpublished manuscript)

George Davida[1] René Peralta[2]

[1] Electrical Engineering and Computer Science Department,
University of Wisconsin-Milwaukee (davida@cs.uwm.edu)
[2] Department of Computer Science
Yale University (peralta-rene@cs.yale.edu)

Abstract. We show that any encryption/decryption system can be converted to an algorithm which uses roughly three machine operations per bit. We prove that our technique yields an encryption/decryption system which is as secure as the original system provided the plaintext has full-entropy. We then show how to remove the full-entropy assumption. Our techniques speed up software implementations of RSA by four orders of magnitude. Alternatively, our techniques can be used as a blueprint for encryption/decryption circuits which are public-key yet are as fast or faster than circuits implementing private-key cryptography.

1 Introduction

In many cryptographic applications, the speed of encryption/decryption is critical. Unfortunately, public-key cryptography is several orders of magnitude slower than secret-key cryptography. For example, many people who “trust” RSA more than DES, are still forced to use the slower RSA only for exchanging a secret key to be used with the faster DES. The same holds for PGP, perhaps the most widely used cryptosystem these days. Furthermore, there are applications for which public-key cryptosystems simply can not be replaced by secret-key cryptosystems.

Some of the efforts in speeding up cryptosystems have involved improvements in the software implementation (see, for example, [4, 7]). In the case of RSA, significant improvements in speed have been hard to come by, mainly because the underlying algorithms are part of a long-established field and hence have already been optimized over many decades of work.[1]

In this paper we show that any encryption/decryption system can be converted to an algorithm which uses roughly three machine operations per bit without compromising security.

2 The cryptosystem

We start with a public-key encryption system E, D over $GF(2)^n$. We will show a construction for encrypting and decrypting full-entropy blocks of size $b >> n$. All vectors and matrices in this paper

[1] Montgomery Multiplication and the many works on addition-chain heuristics for fast modular exponentiation are among the more significant developments in this regard.

are over $GF(2)$. For any vector V, we denote the k^{th} bit of V by v_k. For any matrix A, we denote by A_{ij} the bit in the i^{th} row and j^{th} column of A.

Let $\{R_1, R_2, ...R_n\}$ be a set of row vectors chosen uniformly at random in GF(2) ^{b}. These strings are *public* and form an $n\ x\ b$ matrix R. To encrypt a full-entropy block M of length b, the sender selects a random row-vector J of size n.[2] The sender then computes and sends $F(M) = J\ R + M$ and $E(J)$. To obtain M, the receiver first computes $J = D(E(J))$. Then $M = F(M) + J\ R$.

Note that each block encryption requires a different string J. If the cryptanalyst correctly guesses the string J then he/she has decrypted the block corresponding to that string. However, this has *no implications* about the security of other blocks. Thus $n = \mid J \mid$ should be moderately large. We suggest $n = 128$ for easy implementation on standard architectures.

3 Security Properties

In this section we show some of the provable security properties of our scheme. Our treatment of security is non-asymptotic. That is, we assume the availability of an oracle that breaks our scheme and then we count the number of calls to this oracle which are enough to invert the underlying encyphering function E. For simplicity of exposition, we assume the oracle is deterministic, i.e. it is simply a look-up table.

We consider two models of attack. The first is what we call the non-uniform model. In this model we allow the oracle to be dependent on the matrix R. In this way we are able to show security properties which hold for (almost) all R. The second model of attack is the uniform model. We assume the availability of an oracle that extracts some information about the encrypted message M depending on what R and M are. Note that a given security property is stronger under the non-uniform model of attack than under the uniform model of attack.

3.1 Security in the non-uniform attack model

Here we assume R is fixed. Since R is randomly generated and $b >> n$, we can assume that it possesses full rank (i.e. rank n). It is not hard to show that breaking the underlying cryptosystem is not significantly harder than obtaining M.

Theorem 1. *Fix J, R such that $rank(R) = n$. Suppose there is an oracle Ψ which, on input $(E(J), J\ R + M)$ outputs α such that $PROB.(\alpha = M) = \epsilon$. The probability distribution is taken over uniformly random M. Then an expected number ϵ^{-1} of calls to Ψ are sufficient to obtain J.*

Proof.

The following algorithm can be used to obtain J:

[2] In this paper we will assume that generation of random (or pseudorandom) numbers is cheap, i.e. it can be done at some constant cost per bit. Note, however, that this is not a crucial assumption since we generate n random bits for every $b >> n$ bits of plaintext.

1. Create a random b-bit vector U.
2. Obtain $\Psi(E(J), R, U) = \theta$ (note that θ exists because the function $f(\theta) = J\ R + \theta$ is a bijection on GF(2) ^{b}).
3. Solve the linear equation $X\ R + \theta = U$ for X.
4. If $E(X) = E(J)$ then $X = J$, otherwise go to step 1.

The expected number of iterations is ϵ^{-1}. $\square$

Theorem 1 can be easily strengthened to the following

Theorem 2. *Fix J, R such that rank$(R) = n$. Let J be a set of n column indexes of R such that the columns are linearly independent. For any b-bit string M define M_J as the substring of M defined by J in the obvious way. Suppose there is an oracle Ψ which, on input $(E(J), J\ R + M)$ outputs α such that $PROB.(\alpha = M_J) = \epsilon$. The probability distribution is taken over uniformly random M. Then an expected number ϵ^{-1} of calls to Ψ are sufficient to obtain J from $E(J)$.*

The proof is essentially identical to the proof of Theorem 1. $\square$

Finally, we note that the ability to obtain partial information about any given bit of M implies the ability to fully compute that bit. Let us say that an oracle has an "ϵ advantage" in guessing the value of a binary function $f(x)$ if it returns the correct value for at least $\frac{1+2\epsilon}{2}$ of all inputs x.

Theorem 3. *For any (R, J), an oracle which has an ϵ advantage in guessing any one bit m_k of M on input ($E(J), J\ R + M$) can be used to obtain m_k with virtual certainty. The probability of error after s calls to the oracle is exponentially small in s.*

Proof.

Queries to the oracle can be randomized via the input $(E(J), JR + M + U)$, where U is a uniformly random vector. The vector $M + U$ hides, information theoretically, M from the oracle. Majority decoding is used to obtain, with virtual certainty, the value of the bit. The Chernoff bound

$$Pr.[X > (1+\delta)\mu] < e^{\mu(\delta - (1+\delta)\ln(1+\delta))}$$

applies where

- X is the number of times the oracle returns a $\underline{\text{wrong}}$ answer
- $\mu = s(1/2 - \epsilon)$
- $\delta > 0$.

Thus, letting $\delta = \frac{2\epsilon}{1-2\epsilon}$, we have

$$\begin{aligned} Pr.[\text{majority decoding fails}] &= Pr.[X > s/2] \\ &= Pr.[X > \mu + s\epsilon] \\ &= Pr.[X > (1+\delta)\mu] \end{aligned}$$

$$
\begin{aligned}
&< e^{\mu(\delta-(1+\delta)\ln(1+\delta))} \\
&= \left[e^{\epsilon+0.5\ln(1-2\epsilon)}\right]^{s} \\
&= c^{s}
\end{aligned}
$$

where $c < 1$ (the reader can verify that $c \approx e^{-\epsilon^2/(1-2\epsilon)}$). □

3.2 Security in the uniform attack model

Our next goal is to prove that if any one bit of M is secure then all bits of M are secure. A moment's thought will convince the reader that this is not true in the non-uniform model of attack: some matrices R can "leak" bits of M. For example, any column of R containing only zeroes will expose the bit of M corresponding to that column. We can, however, prove that the number of pairs $(R, JR+M)$ which leak any one bit is statistically insignificant unless the underlying cryptosystem can be broken.

Theorem 4. *Fix J, let $n \leq b$, and let Ω be the uniform probability space consisting of pairs (R, M) where R is an n x b matrix of rank n and M is a vector of size b. Suppose oracle BIT_k takes as input a pair of the form $(R, JR+M)$ and returns m_k with probability $\epsilon > 0$ and "don't know" with probability $1-\epsilon$. Then for all $(R, M) \in \Omega$, J (and hence M) can be recovered from $(R, JR+M)$ with an expected $(b+O(1))\epsilon^{-1}$ number of queries to BIT_k.*

Proof. We randomize the oracle queries as follows. Let T be a b x b matrix chosen uniformly at random. Let U be a vector of length b chosen uniformly at random. Consider the pair $(RT, (JR+M)T+U) = (JR'+M')$, where $R' = RT$ and $M' = MT+U$. Clearly M' is uniformly distributed in $GF(2)^b$ and independent of R'. Since R has full rank, R' is uniformly distributed in $GF(2)^{b^2}$. If we ignore those pairs for which R' does not have full rank, the remaining pairs are uniformly distributed over Ω. Let us say that an oracle query of the form $BIT_k(R', JR'+M')$ is "successful" if it returns m'_k. Each successful call to the oracle yields the linear equation $m'_k = \sum_{i=1}^{b}(m_i T_{ik} + u_k)$. Obtaining b linearly independent such equations is enough to solve for all m_i and hence obtain M. The probability that b linearly independent equations are not yet obtained after $b+s$ successful calls to the oracle is exponentially small in s. The expected number of queries necessary for $b+s$ of them to be successful is $(b+s)\epsilon^{-1}$. □

4 Asymptotic cost of encryption and decryption

We are mainly interested in the per-bit time cost of our scheme. Since decryption costs are essentially the same as those of encryption, we will only discuss the latter. This cost will be dominated by the time necessary to compute $E(J)$ and JR. The per-bit cost is then given by

$$\frac{Time(E(J)) + Time(JR)}{b}. \tag{1}$$

Fixing the parameter n and allowing b to grow, this expression is asymptotic to $Time(JR))/b$. It takes roughly three machine instructions to perform each basic operation involved in computing JR. Thus the expected number of such operations is $(3\ (n/2)(b/w))/b \quad = \quad 3n/2w$, where w is the word length of the machine and $n/2$ is the expected Hamming weight of J. We have proposed using $n = 128$. We may take the value of w to be 64. Thus the asymptotic cost per bit reduces to 3 machine operations. This in turn translates to encryption/decryption speeds on the order of tens of megabytes per second. However, we must note that the memory requirements increase with b. At $b = 2^{21}$, the matrix R is of size $32MB$. The fact that we may not, in practice, allow b to grow arbitrarily may cause the term $Time(E(J))/b$ to dominate in (1).

We implemented our method using $RSA - 1024$ as our underlying encryption function and 32 megabytes of memory. We used one RSA encryption to send $8 = 1024/128$ vectors $J_1 \dots J_8$. Nevertheless, $Time(E(J_i))$ dominates over $Time(J_iR))$. Thus it becomes clear that recursive use of our technique can further speed up encryption/decryption. For example, to encrypt 2^{32} bytes, a single $MASTER\_J$ of length 2^{18} bytes can contain 2^{14} indexes $J_1 \dots J_{2^{14}}$ of length 128 each. $MASTER\_J$ is a random (and therefore full-entropy) string which can be sent using our technique. Only one RSA encryption (of which only the last 128 bits of the RSA encrypted message would be used) is needed. Thus we have managed to encrypt four gigabytes of data using a single RSA encryption. This achieves rates in the tens or hundreds of megabytes per second on a modern PC.

An alternative use of the recursive method is to achieve high speeds while using a much smaller block length (and hence a much smaller matrix R). As an example, this technique yields (on our $150MHz$ processor) encryption speeds of

- $2MB$ per second using a matrix R of size $1MB$;
- $8MB$ per second using a matrix R of size $2MB$.

5 On the full-entropy assumption

Almost all security proofs for cryptosystems rely on some version of the full-entropy assumption (the notable exception being those systems based on probabilistic encryption [5]). However, cleartext is not likely to have the full-entropy property. The applications engineer is often encouraged to use compression to lower the potential risks associated with a cryptanalyst possessing partial information about the cleartext. This is likely (but is not guaranteed) to increase the ratio of entropy to cleartext length.

Our cryptosystem is, however, critically vulnerable to an attack based on partial information about the cleartext. The reader can easily verify that if the cryptanalyst can obtain n bits out of the b bits in a block of plaintext, then the remaining $b - n$ bits of the block can be obtained by simply solving a system of linear equations. To solve this problem, we use a technique of “randomized scrambling”.

6 Randomized scrambling

The technique we use here is a special case of a commonly-used heuristic (what Bellare and Rogaway call "simple-embedding scheme" [3]). The heuristic is used to convert deterministic encryption into randomized encryption. In addition to yielding randomized encryption, this technique aims at making prior knowledge of linear relationships between bits of a plaintext M useless to an attacker.

The scheme is the following:

– generate a random DES key K.
– map the message M to $M' = (K, DES_K(M))$.
– transmit M' rather than M.

Recovery of M now involves one application of DES to a block of length b. DES chips can deliver throughputs of five to ten gigabits per second.[3] This means that adding the randomized scrambling step introduces no significant delay to our design.

The above design would be appropriate for the construction of an ultra fast public-key encryption/decryption board. For software implementations, however, DES is not the optimal choice for randomized scrambling. A faster function, such as SEAL [7] can be used in this case.

7 On the generation and distribution of R

Our design is fastest when R is large. Besides the memory limitations on the size of R, we must also consider scenarios where the speed of generation and/or distribution of R is of concern. In practice, R may not be truly random but pseudo-random. The question then arises of which pseudo-random generator to use. When speed of generation of R is not a concern, a cryptographically secure pseudo random number generator should be used. When speed is a concern, then faster generators can be used. In particular, the techniques of Aiello, Rajagopalan, and Venkatesan should be considered [1]. When seed-size is a concern (either because of storage or speed of distribution constraints), R can be an ϵ-biased string [6, 2]. The latter exhibit super-polynomial expansion while preserving provable pseudo-random properties.

8 Remarks

We remark here on some of the salient features and possible enhancements of our design.

– It speeds up <u>any</u> public-key cryptosystem to a limiting speed of about three machine operations per bit. This effectively removes the speed differential between symmetric and asymetric cryptosystems.
– The resulting cryptosystem is still public-key.

[3] In particular, the chip recently developed at Sandia National Labs can achieve this and higher speeds with a key size of 112 bits. We thank Ed Witzke for providing us with this information.

– It removes any cryptographic weakness that might arise from partial information known to the cryptanalyst when using the underlying public-key cryptosystem. This is because our method uses the latter only to encrypt/decrypt a random bit string.
– Provided one believes in the "scrambling" properties of DES (or whatever function is used for randomized scrambling), our design offers a guarantee of bit-wise security akin, in practice, to that of probabilistic encryption.
– Our technique can also be used to speed-up private-key cryptosystems. This might prove useful in practice for software implementations. However, the "randomized scrambling" function utilized must then be chosen so as to not be a bottleneck with respect to speed.
– Our method yielded a speed-up of four orders of magnitude when applied to our software implementation of RSA. Higher speeds can be obtained by recursive use the technique (see section 4).
– The results presented here extend to public-key cryptosystems which are not length preserving, i.e. $E : Z_2^n - > Z_2^m$ and $D : Z_2^m - > Z_2^n$, where $m > n$.
– Theorem 2 can be strengthened further by allowing the n columns of R to be linearly dependent but of rank no smaller than, say, $n - k$. In this case k bits of M must be guessed in order to recover J in the security proof. In this way we obtain the result that almost all sets of n bits of M are provably secure. To quantify "almost all" we need to know the probability that an $n\ x\ n$ random binary matrix will have rank no smaller than $n - k$. Unfortunately, we know of no closed formula for this probability. However, it is not hard to see that the function $f_n(m, r)$ for the probability that a random binary $n\ x\ m$ matrix has rank r satisfies the following recurrence

$$\begin{aligned} 2^n\ f_n(m,r) &= 2^r\ f_n(m-1,r) + (2^n - 2^{r-1})\ f_n(m-1,r-1) \\ f_n(m,r) &= 0 \qquad\qquad r > max(n,m) \\ f_n(m,0) &= 2^{-nm} \end{aligned}$$

For $n = 128$ and $k = 20$, direct computation yields that the probability that a random binary 128 X 128 matrix has rank less than 108 is approximately 6 x 10^{-133}.
– The proven security properties of our design should be sufficient for most cryptographic applications. In those cases where the user requires proof of stronger security properties (e.g. semantic security or non-malleability), the techniques of Bellare and Rogaway [3] can be used in conjunction with ours. Their techniques enhance the security of any system based on trapdoor one-way functions.

9 Acknowledgments

Moti Yung's comments on a first draft of this paper are gratefully acknowledged. We would also like to thank our students Dennis Meilecke and Fethiye Akbulut for their assistance in software development.

References

1. W. Aiello, S. Rajagopalan, and R. Venkatesan. Design of practical and provably good random number generators. *Journal of Algorithms*, 29:358–389, 1998.
2. N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, 3(3):289–304, 1992.
3. M. Bellare and P. Rogaway. Provably secure session key distribution - the three party case. In *Annual Symposium on the Theory of Computing*, pages 57–66. ACM, 1995.
4. H. Everle. A high-speed DES implementation for network applications. In *Advances in Cryptology - Proceedings of CRYPTO 92*, volume 740 of *Lecture Notes in Computer Science*, pages 521–539, 1993.
5. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
6. J. Naor and M. Naor. Small-bias probability spaces: efficient constructions and applications. In *Proceedings of the 22th Annual ACM Symposium on the Theory of Computing*, pages 213–223, 1990. To appear in Siam Journal on Computing.
7. P. Rogaway and D. Coppersmith. A software-optimized encryption algorithm. *Journal of Cryptology*, 11(4):273–287, 1998. US Patent Issued 5,454,039, 9/26/1995.