Exponentiation Speed Up For Diffie-Hellman Key Agreement Protocol

Shay Gueron [1,3] Or Zuk [2,3]
[1] Dept. of Mathematics, University of Haifa, Haifa, Israel (shay@math.haifa.ac.il)
[2] Faculty of Physics, Weizmann Institute of Science, Rehovot, Israel (or.zuk@discretix.com)
[3] Discretix Technologies, Netanya, Israel.

March 21, 2006

***Abstract*— The Diffie-Hellman key agreement protocol is a well known method in which two parties agree on a secret key by mean of public communication. From the computational point of view, the protocol requires, from both paries, to execute two exponentiations in some group, where the secret exponent is chosen, independently, by each user. In environments with limited resources, such as smartcards, these computations are considered heavy.**

In this paper we propose a method for decreasing the computational work involved with this exponentiation. If the required entropy of the secret exponent is K, the straightforward approach is to select this key as a string of K random bits. Instead, we choose here longer keys with imposed limitations on their Hamming weight, in a way that the entropy remains K. We show how this can reduce the exponentiation time by $\sim 4\%$ to $\sim 9\%$, depending on the group. The relative performance gain is shown to be independent of K. Finally, we show how our method can be combined with more sophisticated exponentiation algorithms, yielding smaller, but sometimes still significant extra-gain in performance.

I. Introduction

The Diffie-Hellman key agreement protocol (see e.g., [1]) is a commonly used protocol with which two communicators can agree on a secret key via public communication. The protocol uses a public commutative group $\mathcal{G}$ and a public element $A \in \mathcal{G}$. Communicator I chooses a secret positive integer X (key I), and Communicator II chooses a secret positive integer Y (key II). Communicator I computes $B_I = A^X$ (in the group), and sends B_I (via public communication) to Communicator II who computes B_I^Y. Communicator II computes $B_{II} = A^Y$ and sends B_{II} to Communicator II who computes B_{II}^X. The agreed key is $B_I^Y = B_{II}^X = A^{XY}$. For each communicator, this key agreement protocol involves the computational cost of two exponentiations, where the exponent is chosen by the communicator.

In the classical protocol, $\mathcal{G}$ is the group of residues modulo some (large) prime P, and exponentiation is $A^X \pmod P$. In more efficient implementations $\mathcal{G}$ is an elliptic curve over some finite field, and exponentiation is scalar multiplication of a (fixed) point on the curve.

Since our discussion is independent of the actual group, we denote the multiplication operation of $X, Y \in \mathcal{G}$ by $MUL(X,Y)$. If F is a positive integer, the exponentiation A^F in G is $MUL\,(A, MUL\,(A, MUL\,(A, \ldots)))$ where the multiplications are repeated F times. We denote the time required for computing one MUL operation by M, and the time required for computing one square (i.e., $MUL(X,X)$) by S. For convenience, we also denote the multiplication time to squaring time ratio by $R = M/S$. Clearly, $R \geq 1$.

For a given exponentiation algorithm $\mathcal{A}$, and a probability distribution on a family of exponents $\mathcal{F}$, we denote the associated expected exponentiation time, when the exponent is drawn from $\mathcal{F}$, by $T = T(\mathcal{F}, \mathcal{G}, \mathcal{A})$. Here, we shall focus on the square-and-multiply ($\mathrm{SquareMul}$) exponentiation algorithm:

SquareMul Algorithm
Input: $A \in \mathcal{G}$, F (positive integer
with binary representation $[F_{f-1}, \ldots, F_1, F_0]$.
Output: A^F

1.1 $U_{f-1} = A$
1.2 For i from $f-2$ to 0 do
1.3 1.2 $\quad U_i = MUL(U_{i+1}, U_{i+1})$
1.4 1.3 $\quad$ if $F_i = 1$ then $U_i = MUL(U_i, A)$
1.5 End For
1.6 Return U_0

Here, if $\mathcal{F}$ is the uniform distribution over the set of f bits integers, the *expected* exponentiation time using the $\mathrm{SquareMul}$ algorithm is $T(\mathcal{F}) = (f-1)S + \frac{1}{2}(f-1)M$.

1) **Defining The Optimization Problem***:* For the key agreement procedure, each communicator chooses his own secret exponent (key), and uses it twice, for two exponentiations. This secret exponent is never revealed, so it can be chosen at will, by communicator, as long as it is safe enough. Suppose that security requirements determine that the entropy of the secret key, $H(KEY)$, equals K. This leads to the following optimization problem.

The Optimization Problem*:* Minimize $T(\mathcal{F}, \mathcal{G}, \mathcal{A})$,
s.t. $H(\mathcal{F}) = K$.

2) **The reference point** $T_{\frac{1}{2}}$*:* The straightforward way to generate a key with $H(KEY) = K$, is to select a random string of K bits. When the $\mathrm{SquareMul}$ algorithm is used, with such a key, the expected exponentiation time is

$$T_{\frac{1}{2}} \equiv (K-1)S + \frac{1}{2}KM \approx KS(1 + \frac{1}{2}R). \qquad (1)$$

(the latter approximation assumes that K is sufficiently large). We consider the exponentiation time $T_{\frac{1}{2}}$ as a reference point.

3) **The Goal***:* We illustrate here a key selection procedure having key entropy K, and achieving exponentiation time smaller than $T_{\frac{1}{2}}$. To achieve this, we select keys with more than K bits which are drawn from an unbalanced bits distribution. The efficiency of the method depends only on the group (through the parameter R), but not on K.

II. The Key Selection Procedure

The proposed key selection procedure uses two parameters n and p. We select a string of n bits, which are drawn, independently, from a binomial distribution having probability $p \in (0,1)$ for a bit to be 1, in a way that the resulting key entropy is K. To this end, (n,p) must should satisfy

$$K = H(KEY) = H(n,p) = \\ -n\left(p\log_2(p) + (1-p)\log_2(1-p)\right). \tag{2}$$

The expected number of 1 bits in this key is np. If the $\mathrm{SquareMul}$ algorithm is used, the expected exponentiation time is (closely)

$$T = T(p,n) = nS + npM = nS(1+Rp). \tag{3}$$

Note that the reference point $T_{\frac{1}{2}}$ is the special case $T(\frac{1}{2}, K)$.

Consequently, using Eq. (2), the optimization problem reduces to minimizing the one-variable function

$$T(p) = \frac{-KS(1+Rp)}{p\log_2(p) + (1-p)\log_2(1-p)}, \quad 0 < p < 1. \tag{4}$$

It can be verified (we skip the computations) that $T(p)$ is convex in $(0,1)$. Its unique minimum in $(0,1)$ is obtained at the unique point p^* satisfying the equation

$$p^* = (1-p^*)^{R+1}, \quad p^* \in (0,1). \tag{5}$$

Clearly, $p^* \neq \frac{1}{2}$ for any $R > 0$, which indicates that we can indeed achieve exponentiation time shorter than $T_{\frac{1}{2}}$. Figure 1 plots the value of of p^* as a function of R. As expected, p^* decreases monotonically with R because larger R implies that multiplication is more and more costly, so it pays to have less 1 bits in the key.

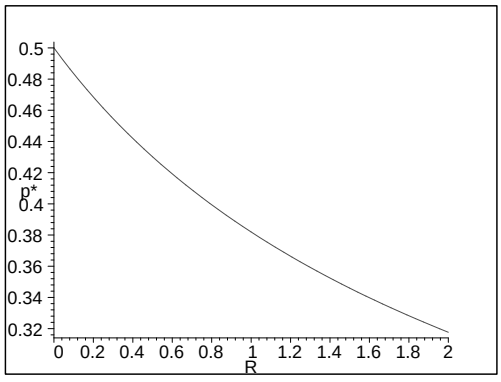

Fig. 1. The value of p^* as a function of R.

At the minimizing point p^* we compute the resulting key length (n^*), which is larger than K, and the optimal exponentiation time (T^*), which is smaller than $T_{\frac{1}{2}}$. We get

$$n^* = -K\frac{R+1}{(1+Rp^*)\log_2(p^*)}, \\ T^* = \frac{-KS(1+R)}{\log_2(p^*)} = \frac{-KS}{\log_2(1-p^*)}. \tag{6}$$

As expected, to maintain the required entropy, the key length (for any $p \neq \frac{1}{2}$, including $p = p^*$) must be increased. Figure 2 shows the relative increase of the key length, at the minimization point $p = p^*$, compared to the length when $p = \frac{1}{2}$, as a function of R.

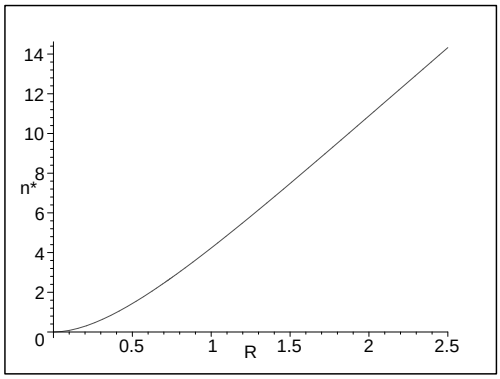

Fig. 2. The relative increase of the key length (in %), compared to K (the key length when $p = \frac{1}{2}$), at the minimization point $p = p^*$, as a function of R.

Most important, the relative gain in the execution time, at the optimum point p^*, is

$$gain^*(R) = 1 + \frac{2}{(2+R)\log_2(1-p^*)}. \tag{7}$$

Note that both p^*, and the relative gain depend only on R, and are independent of the required key entropy K. Figure 3 plots the relative gain in the exponentiation time, at the optimum point p^*, as a function R. It is not surprising to realize that this gain is an increasing function of R, because our approach actually replaces "expensive" multiplications by a few more squares that result in from using a longer key.

Examples: The Diffie-Helman protocol is used in practice with three groups: $\mathbb{Z}_P^*$, Elliptic curves (EC) over $GF(P)$ (for some large prime P) and Elliptic curves over $GF(2^k)$. Each group has an associated value of R (depending also on the actual details of the implementation). Table I summarizes the parameters related to the proposed method, when applied to these specific groups. The second column of Table 1 shows the value of R, where the EC values are based on the usage of mixed Jacobian coordinates (see [2] for more details). The other columns show the optimal probability p^*, the Keylength increase, and the relative gain in performance. As can be seen, depending on the group that is used, a 4% to 9% gain in the performance can be expected.

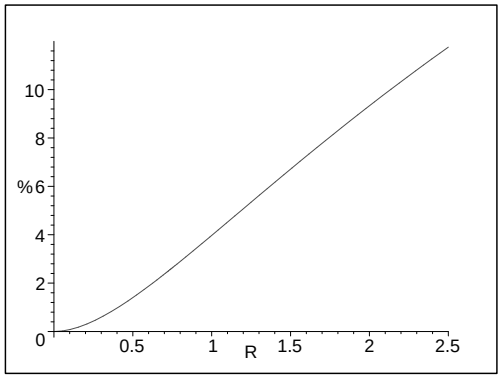

Fig. 3. The relative gain (in $\%$) in the exponentiation time, at the optimum point p^*, as a function R, in the range $0 < R < 2.5$.

Group $\mathcal{G}$	R	p^*	Keylength increase (in $\%$)	Relative Gain (in $\%$)
DH over $\mathbb{Z}_P^*$	1	0.38	4.23	3.97
EC over $GF[P]$	1.4	0.352	6.8	6.2
EC over $GF[2^n]$	2	0.318	11	9.3

TABLE I

THE VALUES OF R, p^*, THE RELATIVE INCREASE IN THE KEY LENGTH, AND THE RELATIVE GAIN IN EXPONENTIATION TIME (IN $\%$), OBTAINED WHEN THE PROPOSED PROCEDURE IS APPLIED. THE DETAILS ARE GIVEN FOR THREE USEFUL GROUPS, AS EXPLAINED IN THE TEXT.

III. IMPLEMENTATION

To apply the proposed method in practice, we need a source of biased bits, according to the relevant value of p^*. To this end, we illustrate here a simple method for obtaining satisfactory biased bits from a balanced source.

Suppose that the system (e.g., smartcard) has some random bits generator (typically a smartcard would have hardware-generated short seed, and a PRNG algorithm that produces). We approximate p^* by fraction of the type $\frac{a}{2^\tau}$. Then, we use τ consecutive bits from the (balanced) PRNG source, to define an integer t. We extract a (single) bit whose value is 1 if $t \leq a$, and 0 otherwise. The quality of the approximation is determined by τ, which determines how many bits are required in order to generate one biased bit.

Example: Consider the classical Diffie-Hellman protocol where modular exponentiation is used. In this case, case $R = 1$, and (see Table 1) $p^* = 0.382$. We use $a = 3$ and $\tau = 3$ to approximate $p^* = 0.382 \approx 3/8$. This means that we need to draw chunks of 3 bits and extract one bit from each chunk. This bit is 1 if the three bits are 000, 001, or 010, and 0 otherwise.

Of course, using this approximation, the algorithm is only near the optimal point p^*. To realize the effect of missing the optimal point, we need some stability analysis for the performance, near p^*. The expression for the relative gain, as a function of p is

$$gain(p) = \frac{T_{1/2} - T(p)}{T_{1/2}} = 1 + \frac{2(1+Rp)}{(2+R)(p\log_2(p) + (1-p)\log_2(1-p))}. \quad (8)$$

Thus, if we use $p = 0.375$ instead of $p^* = 0.382$, the relative gain drops from $\sim 3.97\%$ to $\sim 3.96\%$. Fortunately, only a slight penalty.

Figure 4 plots the relative gain in the exponentiation time (in $\%$), as a function of p, for the three cases of interest: $R = 1,\ 1.4$, and 2. For each curve, the peak is at p^*, and the figure helps estimate the effect of choosing an approximate value of p instead. Note that the three curves cross the horizontal axis at some point, implying that if p is too far form the optimal point, the resulting exponentiation time is actually worst than $T_{\frac{1}{2}}$.

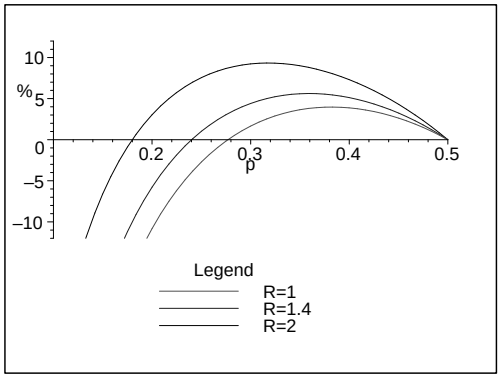

Fig. 4. The relative gain in the exponentiation time (in $\%$) as a function p, for three representative values of R: $R = 1$, $R = 1.4$, and $R = 2$.

IV. OTHER EXPONENTIATION ALGORITHMS

Up to this point the analysis was performed under the assumption that exponentiation is carried out by using the SquareMul algorithm. This analysis needs to be modified when other exponentiation algorithms are used. Qualitatively, it is clear that the efficiency of our proposed method decreases if the relative weight of the multiplication operations decreases. To illustrate, this section analyzes the Sliding Windows (SW) exponentiation algorithm (see, for example [3] and details in [1]).

For $p = \frac{1}{2}$, using a window of width k, the average number of MUL operations is $2^{k-1} - 1 + \frac{n-1}{k+1}$. In practice, due memory limitations k is chosen to be small (typically 2 to 4), and this justifies approximating the number of multiplications by (for large n) by $\sim \frac{n}{k+1}$ (to be compared with $\frac{n}{2}$ multiplications required with the SquareMul algorithm).

We now compute the number of multiplications associated with the SW algorithm, when using a binomial distribution for the key bits (with probability p for a bit to be 1). Recall that in the SW algorithm, we perform one multiplication for every window (of size k). After finishing a window, we "open" the next window upon encountering the first 1-bit. The expected number of zeros before this 1-bit is $\frac{1-p}{p}$. Thus, on

average, we perform one multiplication per every $k+\frac{1-p}{p}$ bits. Consequently, the expected total number of multiplications is

$$2^{k-1}-1+\frac{(n-1)p}{kp+1-p}\approx\frac{np}{kp+1-p}, \tag{9}$$

and the expected exponentiation time is

$$T_{SW}(p)=\frac{-KS(1+p(k+R-1))}{(kp+1-p)(p\log_2(p)+(1-p)\log_2(1-p))}. \tag{10}$$

Note that for a trivial window, i.e., $k=1$, these equations reduce to those that represent the $\mathrm{SquareMul}$ algorithm analyzed above. The optimal value of p, denoted p^*_{SW}, is the unique solution of the equation

$$\left(k^2+kR-2\,k-R+1\right)p^2+(2\,k-2)\,p+1=\frac{R\log_2(1-p)}{\log_2(p)-\log_2(1-p)},\quad p\in(0,1). \tag{11}$$

Table II shows the results obtained by using the proposed method, with the three groups, using three window lengths ($k=2,3,4$).The table gives the optimal value of p, p^*_{SW}, and the relative gain in performance. As expected, the gain is indeed less appreciated, especially as the window length increases. Our conclusion is that the proposed method is appropriate for $k=2$ and for the EC groups.

Group $\mathcal{G}$	R	k	p^*	Relative Gain (in %)
DH over $\mathbb{Z}^*_P$	1	2	0.437	1.06
DH over $\mathbb{Z}^*_P$	1	3	0.462	0.375
DH over $\mathbb{Z}^*_P$	1	4	0.475	0.164
EC over $GF[P]$	1.4	2	0.417	1.77
EC over $GF[P]$	1.4	3	0.45	0.65
EC over $GF[P]$	1.4	4	0.467	0.29
EC over $GF[2^n]$	2	2	0.391	2.91
EC over $GF[2^n]$	2	3	0.433	1.11
EC over $GF[2^n]$	2	4	0.456	0.5

TABLE II

THE RESULTS OBTAINED BY USING THE PROPOSED METHOD WITH THE THREE GROUPS, USING THE SW ALGORITHM WITH WINDOW LENGTHS $k=2,3,4$. THE OPTIMAL VALUE OF p, p^*_{SW}, AND THE RELATIVE GAIN IN PERFORMANCE (IN %) ARE SHOWN. AS EXPECTED, THE GAIN DROPS SIGNIFICANTLY WITH THE WINDOW LENGTH INCREASES. THE PROPOSED METHOD IS APPROPRIATE ONLY FOR $k=2$ AND THE EC GROUPS.

V. CONCLUSION

We have demonstrated a method for decreasing the computational cost of the two exponentiations involved with the Diffie-Hellman key agreement protocol.

The underlying idea is to use keys that are longer than K, the required entropy, but to draw the bits from some biased binomial distribution. This offers a tradeoff between an increased number of squares and a decreased number of (more expensive) multiplication. By nature, this method is more successful in case where squaring is significantly cheaper than multiplication. The expected improvement depends on the group that is being used, and on the exponentiation algorithm, and is independent of K.

To use the method, we first need to compute the optimal probability for a 1-bit p^*, and the extended key length, n^*. When this method is used together with the commonly used $\mathrm{SquareMul}$ exponentiation algorithm, the relative performance gain varies from $\sim 4\%$ (for $\mathbb{Z}_P$) to $\sim 9\%$ (for $GF(2^k)$. No additional memory requirements are involved, except for a slight increase in the Keylength.

In practice, the optimal value p^* can be approximated by fraction of the type $\frac{a}{2^\tau}$, for some τ (e.g., $p^*=0.382\approx 3/8$, using $a=3$ and $\tau=3$). To generate the biased bits, τ balanced bits are drawn, defining an integer t, and a single bit is then extracted: 1 if $t\leq a$., and 0 otherwise. It was shown that the deviation from the optimum point has only a small effect on the exponentiation time improvement.

Finally, we showed how other exponentiation algorithms can be analyzed. In the example of the SW algorithm, the analysis shows that the proposed method is still useful, but the contribution is distinguishable only with small windows (which require less memory), and on EC groups.

REFERENCES

[1] Menezes, A. J. Oorschot, P.C., and Vanstone,S.A.: Handbook of Applied Cryptography. CRC Press, New York (1997).

[2] I. Blake, G. Seroussi and N. Smart, Elliptic Curves in Cryptography. London Mathematical Society Lecture Notes Series # 265. Cambridge University Press, Cambridge 1999.

[3] E.G. Thurber, On addition chains $l(mn)\leq l(n)-b$ and lower bounds for $c(r)$. Duke Mathematical Journal 40 (1973), 907913.