Minimal Weight k-SR Representations

Yongfei Han, Dieter Gollmann, Chris Mitchell

Information Security Group
Department of Computer Science
Royal Holloway, University of London
Egham
SurreyTW20 0EX
E-mail: {yongfei, dieter, cjm@dcs.rhbnc.ac.uk}

Abstract. An algorithm for a minimal weight string replacement representation for the standard square and multiply exponentiation method is discussed, with a presentation of the design and proof of the algorithm. The performance of this new method is analysed and compared with previously proposed methods. The techniques presented in this paper have applications in speeding up the implementation of public-key cryptographic algorithms such as RSA [3].

1 Introduction

We are investigating the property of a redundant integer representation which can help in reducing the number of multiplications in a square & multiply exponentiation by reducing the weight of exponents [1]. The concept of k-SR representations is based on the idea of replacing 1-runs in the binary representation of an integer by a single digit. We consider runs up to length k, hence the name k string replacement. An obvious conversion algorithm would just substitute 1-runs by k-SR digits. However, it was observed that this does not necessarily lead to optimal results. For example, the number $21 = (10101)_2$ contains three 1-runs but has a 3-SR representation of weight 2, i.e. (77). In this paper, we will examine algorithms for computing (nearly) minimal k-SR representations.

Definition 1. k-SR representation: A k-SR (string replacement) representation of an integer e is given by

$$e = \sum_{i=0}^{r} f_i 2^i, \quad \text{with } f_i = 2^{k_i} - 1,\ 0 \le k_i \le k.$$

Note that k-SR representations are not unique, e.g. we have the following 4-SR representations of 21: 10101, 77, $3F$. (F is used to represent 15.) In the following, let $\underline{x}$ be a k-SR number.

Definition 2. The number of 0-runs in $\underline{x}$ is written as $r_0(\underline{x})$. The number of 1-runs in $\underline{x}$ is written as $r_1(\underline{x})$.

Definition 3. Weight: The *weight* of a string $\underline{x}$, written as $w(\underline{x})$, is defined as the number of non-zero symbols in $\underline{x}$. With $w_i(\underline{x})$, we denote the length of the i-th 1-run in $\underline{x}$.

2 Staggered k-SR Representations

In this Section, we will define a standard representation of k-SR numbers and show that we always can find a minimal k-SR representation of this form. These properties can then be the foundation for developing a minimisation algorithm. Let e have a minimal representation

$$e = \sum_{i=0}^{r} (2^{j_i} - 1)2^{l_i}.$$

Note that we have by definition $l_i < l_{i+1}$ for all $i < r$.

Definition 4. A k-SR number $\sum_{i=0}^{r}(2^{j_i} - 1)2^{l_i}$ is called *staggered* if we have $j_i + l_i < j_{i+1} + l_{i+1}$ for all $i < r$.

Lemma 5. *Every integer e has a staggered k-SR representation of minimal weight.*

Proof: Assume we have a minimal k-SR representation with $j_i + l_i \geq j_{i+1} + l_{i+1}$ for some i. We rewrite

$$(2^{j_{i+1}} - 1)2^{l_{i+1}} + (2^{j_i} - 1)2^{l_i} = (2^{j_i + l_i - l_{i+1}} - 1)2^{l_{i+1}} + (2^{j_{i+1} + l_{i+1} - l_i} - 1)2^{l_i}$$

and get a valid k-SR representation as

$$j_i > j_i + l_i - l_{i+1} \geq j_i + l_i - j_{i+1} - l_{i+1} \geq 0, \quad j_i \geq j_{i+1} + l_{i+1} - l_i \geq l_{i+1} - l_i > 0.$$

□

Theorem 6. *In a minimal k-SR representation, $j_0 - l_0$ is the position of a 0 terminating a 1-run or to a 1 in a 1-run of length strictly greater than k.*

Proof: Assume that we can find a minimal staggered k-SR representation where our assumption does not hold. Without loss of generality, assume $l_0 = 0$. First, assume that j_0 points to some other 0 in e, e.g. $j_0 = 4$ in $e = \ldots 00101$.

$$\begin{array}{r} 1\,1\,1\,1 \\ \ldots 1\,1\,1\,1 \\ \hline \ldots 0\,\underline{1}\,1\,0\,1 \end{array}$$

Because we have a staggered representation we will always have a 1 in position $j_0 - 1$ unless we add a correcting digit $2^{j_1} - 1$ in this position. We get an alternative representation of equal weight and reduced j_0 as

$$(2^{j_0} - 1) + (2^{j_1} - 1)2^{j_0 - 1} = 2^{j_1 + j_0 - 1} + (2^{j_0 - 1} - 1).$$

Applying this step repeatedly, we can make j_0 point to the terminator of a 1-run. In our second case, we let j_0 point to some 1 in e, e.g. $j_0 = 5$ in $e = \ldots 110101$.

$$\begin{array}{r} 1\,1\,1\,1\,1 \\ \ldots 1\,1\,1\,1\,1 \\ \hline \ldots \underline{0}\,1\,1\,1\,0\,1 \end{array}$$

Because we have a staggered representation we will always have a 0 in position j_0 unless we add a correcting digit $2^{j_2}-1$ in this position. Assume we also have to correct a 0 in a position $l_1 < j_0$ by adding $(2^{j_1}-1)2^{l_1}$. We get an alternative representation of equal weight and reduced j_0 as

$$(2^{j_0}-1)+(2^{j_2}-1)2^{j_0}+(2^{j_1}-1)2^{l_1} = 2^{j_2+j_0}+(2^{j_1-1}-1)2^{l_1+1}+(2^{l_1}-1).$$

Applying this step repeatedly, we can make j_0 point to the terminator of a 1-run. If we do not have to correct a 0, we get can increment j_0 as

$$(2^{j_0}-1)+(2^{j_2}-1)2^{j_0} = 2^{j_2+j_0}-1 = (2^{j_0+1}-1)+(2^{j_2-1}-1)2^{j_0+1}.$$

□

In the following, we show how the choice of the l_i can be restricted in a minimisation algorithm. We will frequently make use of the fact that we may construct a staggered representation. Adding a new digit $(2^{j_i}-1)2^{l_i}$ to an intermediate result will have the following effect:

- When the intermediate result has a 1 in position l_i, then $(2^{j_i}-1)2^{l_i}$ is a *0-corrector*, changing this bit to 0, leaving the remainder of the intermediate result unchanged, and appending a string $10\ldots0$ at the top.
- When the intermediate result has a 0 in position l_i, then $(2^{j_i}-1)2^{l_i}$ is a *1-corrector*, changing any string $10\ldots0$ starting in position l_i to $01\ldots1$ leaving the remainder of the intermediate result unchanged, and appending a string $10\ldots0$ at the top.

Adding $(2^{j_i}-1)2^{l_i}$ to an intermediate result, therefore, will only imply local changes at l_i and the addition of a string at the top.

Lemma 7. *When computing a minimal k-SR representation for a given j_0 and $l_i < j_0+l_0$, then j_i+l_i cannot be the position of a 0 in the 'first available' 0-run.*

Proof: W.l.o.g. assume that j_1+l_1 is the position of a 0 in the first 0-run above j_0, e.g. $j_1=3$ and $l_1=1$ in $e=1001101$.

$$\begin{array}{r} 1\,1\,1\,1 \\ 1\,1\,1\,1 \\ \hline \underline{1}\,0\,1\,1\,0\,1 \end{array}$$

Adding $(2^{j_1}-1)2^{l_1}$ to the intermediate result will generate a 1 in position j_1+l_1. We do not have to correct any 1 in a position below j_1+l_1. We may have to correct some 0's but because we are constructing a staggered representation, no 0-corrector would change the 1 in position j_1+l_1. Therefore, we have to add a corrector $(2^{j_2}-1)2^{j_1+l_1}$. We get an alternative representation of equal weight for the digits in position l_0, l_1, l_2 which meets the condition of our Lemma as

$$\begin{aligned}&(2^{j_0}-1)2^{l_0}+(2^{j_1}-1)2^{l_1}+(2^{j_2}-1)2^{j_1+l_1}\\ &\quad=(2^{l_1-l_0}-1)2^{l_0}+(2^{j_0+l_0-l_1-1}-1)2^{l_1+1}+2^{j_1+j_2+l_1}.\end{aligned}$$

□

Lemma 8. *When computing a minimal k-SR representation for a given j_0 and $l_i < j_0 + l_0$, then $j_i + l_i$ can be the position of a 1 only if this 1 is the terminator of a 0-run.*

Proof: W.l.o.g. assume $j_1 + l_1$ is the second position in a 1-run, e.g. $j_1 = 4$ and $l_1 = 1$ in $e = \ldots 110101$.

$$\begin{array}{r} \quad 1\,1\,1 \\ \quad 1\,1\,1\,1 \\ \hline \ldots 1\,\underline{0}\,0\,1\,0\,1 \\ \ldots 1\,1 \\ \hline \ldots \underline{0}\,1\,0\,1\,0\,1 \end{array}$$

Adding $(2^{j_1} - 1)2^{l_1}$ to the intermediate result will generate a 0 in the position where the 1-run starts. Adding a digit $(2^{j_2} - 1)2^{l_2}$ to correct this 0 will change the 1 in position $j_1 + l_1$ to a 0 and we have to add another digit $(2^{j_3} - 1)2^{j_1+l_1}$. Note that other digits in a staggered representation will not affect these positions. We get an alternative representation of equal weight for these digits which meets the condition of our Lemma as

$$\begin{aligned}(2^{j_1} - 1)2^{l_1} + (2^{j_2} - 1)2^{l_2} + (2^{j_3} - 1)2^{j_1+l_1} \\ = (2^{l_2-l_1} - 1)2^{l_1} + (2^{j_2-1} - 1)2^{l_2+1} + 2^{j_1+j_3+l_1}.\end{aligned}$$

□

Lemma 9. *When computing a minimal k-SR representation for a given j_0 and $l_i < j_0 + l_0$, then $j_i + l_i$ can be the position of a 0 only if it this 0 is the terminator of a 1-run.*

Proof: W.l.o.g. assume that l_2 is the start of a 1-run, l_3 is the start of the next 1-run, and $l_2 < l_3 < j_1 + l_1$, e.g. $l_2 = 3$, $l_3 = 5$, and $j_1 + l_1 = 6$ in $e = \ldots 00110101$.

$$\begin{array}{r} \quad 1\,1\,1 \\ \quad 1\,1\,1\,1\,1\,1 \\ \hline \ldots 1\,0\,0\,\underline{0}\,0\,1\,0\,1 \\ \ldots 1\,1\,1\,1 \\ \hline \ldots 0\,\underline{1}\,1\,1\,0\,1\,0\,1 \end{array}$$

Adding $(2^{j_1} - 1)2^{l_1}$ to the intermediate result will generate a 0 in position l_2. Adding a digit $(2^{j_2} - 1)2^{l_2}$ to correct this 0 will create 1's in positions l_3 to $j_1 + l_1 - 1$. We have to correct these 1's individually. Consider, for example, position $j_1 + l_1 - 1$. We add a digit $(2^{j_4} - 1)2^{j_1+l_1-1}$ but

$$(2^{j_1} - 1)2^{l_1} + (2^{j_4} - 1)2^{j_1+l_1-1} = (2^{j_1-1} - 1)2^{l_1} + 2^{j_1+j_4+l_1-1}.$$

Applying this argument repeatedly, we get a representation of equal weight and $j_1 + l_1 = l_3$.

□

Lemma 10. *When computing a minimal k-SR representation for a given j_0 and $l_i < j_0 + l_0$, then $j_i + l_i$ cannot be the terminator of a 1-run of length 1.*

Proof: W.l.o.g. assume that $j_1 + l_1$ is the terminator of a 1-run of length 1. We have to add a 1-compensator in position $j_1 + l_1 - 1$ and a 0-compensator in position $j_1 + l_1$, but

$$\begin{aligned}&(2^{j_1} - 1)2^{l_1} + (2^{j_2} - 1)2^{j_1+l_1-1} + (2^{j_3} - 1)2^{j_1+l_1} \\ &\quad = (2^{j_1-1} - 1)2^{l_1} + (2^{j_2-1} - 1)2^{j_1+l_1} + 2^{j_1+j_3+l_1}.\end{aligned}$$

□

3 Computing a k-SR representation

We now can formulate strategies for finding minimal k-SR representations of an integer e. We start from the binary representation of e. Let l_0 be the position of the least significant 1 in e.

1. Choose $j_0 + l_0$ to be the terminator of a 1-run or $j_0 = k$ if there is no such terminator;
2. For every l_i with $e_{l_i} = 0$ and $l_0 < l_i < j_0$ add a 0-corrector; we correct from the least significant 0 upwards by matching the 'next available 0-run in e'.
3. Strategy 1: choose j_i so that $j_i + l_i$ is the most significant position in this 0-run;
4. Strategy 2: choose j_i so that $j_i + l_i$ is the most significant position in the 1-run following this 0-run; add the least significant position of this 1-run into a list of positions which have to be corrected;
5. If the next available 0-runs is followed by a 1-run of length 1, then choose Strategy 1.

For a given binary string $\underline{x}$ with least significant bit 1, we now compute a k-SR representation using Strategy 1 only.

$$\sum_{i=0}^{r} \left(2^{j_i(\underline{x})} - 1\right) 2^{l_i(\underline{x})}.$$

Let the $l_i(\underline{x})$ be the position of the i-th least significant 0 in $\underline{x}$. Define $j_0(\underline{x}) \leq k$ to be the terminator of a 1-run and let $N_{j_0}(\underline{x})$ be the number of zeroes in $\underline{x}[j_0(\underline{x}), 0]$. For $i > 0$ and $l_i(\underline{x}) < j_0(\underline{x})$, define $j_i(\underline{x})$ iteratively by the algorithm given in Figure 1. For example, with $k = 15$, $\underline{x} = 11\ 0111\ 0101\ 0001\ 0010\ 0101\ 0101$, and $j_0(\underline{x}) = 13$ we get

i	0	1	2	3	4	5	6	7
l_i	0	1	3	5	7	8	10	11
j_i	13	15	15	15	14	14	14	14

and the representation of weight 8,

$$(2^{14} - 1)(2^{14} - 1)0(2^{14} - 1)\ (2^{14} - 1)0(2^{15} - 1)0\ (2^{15} - 1)0(2^{15} - 1)(2^{13} - 1).$$

```
p := j_0(x);        /* p is a pointer */
i := 1;
WHILE l_i(x) < j_0(x) DO
      p := p + 1;
      IF x_p = 1 THEN
         j_i(x) := p − l_i(x);
         IF j_i(x) > k THEN STOP
         i := i + 1;
      END IF
END WHILE
```

Fig. 1. A k-SR replacement algorithm

However, the maximal $j_0(\underline{x})$ is not always the optimal choice, in our example we have also 15-SR representations of weight 7,

$$10\ 0000\ (2^5-1)(2^5-1)00\ 0000\ 0000\ 00(2^{11}-1)0\ (2^9-1)0(2^8-1)(2^7-1)$$

or

$$00\ 0000\ (2^6-1)0(2^7-1)(2^7-1)\ 0001\ 0000\ 0000\ (2^6-1)0(2^5-1)(2^5-1).$$

We now prove that the algorithm given in Figure 1 is correct.

Lemma 11. *Assume that $\underline{x}$ represents an odd number. If $j_i(\underline{x})$, $0 \le i \le r$, are successfully computed for some r, then $\underline{x}$ and the binary expansion of*

$$\tilde{\underline{x}}_r = \sum_{i=0}^{r} (2^{j_i(\underline{x})} - 1)2^{l_i(\underline{x})}$$

will coincide in positions 0 to $l_r(\underline{x})$ and positions $j_0(\underline{x}) - 1$ to $l_r(\underline{x}) + j_r(\underline{x})$. Between positions $l_r(\underline{x})+1$ and $j_0(\underline{x})-1$, the coefficients of the binary expansion of $\tilde{\underline{x}}_r$ are all 1.

Proof (by induction): For $r = 0$, we have $\tilde{\underline{x}}_0 = 2^{j_0(\underline{x})} - 1$ As $\underline{x}$ is odd and $l_0(\underline{x}) = 0$, we have $\underline{x}_0 = 1$, $\underline{x}_{j_0(\underline{x})-1} = 1$, and $\underline{x}_{j_0(\underline{x})} = 0$ so the Lemma holds.

Assume that all the assumptions of the Lemma hold for a given r. The next 0 in $\underline{x}$ will occur in position $l_{r+1}(\underline{x})$. If $l_{r+1}(\underline{x}) \ge j_0(\underline{x})$ we stop and observe that $\underline{x}$ and the binary expansion of $\tilde{\underline{x}}_r$ now coincide in positions 0 to $l_r(\underline{x}) + j_r(\underline{x})$.

If $l_{r+1}(\underline{x}) < j_0(\underline{x})$ and if $j_{r+1}(\underline{x})$ can be computed, then we update $\tilde{\underline{x}}_r$ by adding $(2^{j_{r+1}(\underline{x})} - 1)2^{l_{r+1}(\underline{x})}$. This addition will switch the coefficient of the binary expansion of $\tilde{\underline{x}}_{r+1}$ in position $l_{r+1}(\underline{x})$ to 0 and leave all coefficients in positions $l_{r+1}(\underline{x}) + 1$ to $l_r(\underline{x}) + j_r(\underline{x})$ unchanged. Thus, $\underline{x}$ and the binary expansion of $\tilde{\underline{x}}_{r+1}$ coincide in positions 0 to $l_{r+1}(\underline{x})$ and in positions $j_0(\underline{x})$ to $l_r(\underline{x}) + j_r(\underline{x})$.

Define $\delta := l_{r+1}(\underline{x}) + j_{r+1}(\underline{x}) - (l_r(\underline{x}) + j_r(\underline{x})) - 1$. Note that $\delta = 0$ if $l_{r+1}(\underline{x}) + j_{r+1}(\underline{x}) = l_r(\underline{x}) + j_r(\underline{x}) + 1$ and that δ is the length of a 0-run otherwise. The coefficients of $\tilde{\underline{x}}_{r+1}$ in position $l_r(\underline{x}) + j_r(\underline{x})$ and above are therefore $10^\delta 1$. We have to consider two cases.

1. $\delta = 0$: then the coefficient of $\underline{x}$ in position $l_r(\underline{x}) + j_r(\underline{x}) + 1 = 1$; hence $\underline{x}$ and the binary expansion of $\tilde{\underline{x}}_{r+1}$ coincide also in position $l_{r+1}(\underline{x}) + j_{r+1}(\underline{x}) = l_r(\underline{x}) + j_r(\underline{x}) + 1$.
2. $\delta \neq 0$: there is a 0-run in $\underline{x}$ which is terminated by a 1 in position $l_{r+1}(\underline{x}) + j_{r+1}(\underline{x})$; again $\underline{x}$ and the binary expansion of $\tilde{x}_{r+1}$ coincide in all positions up to $l_{r+1}(\underline{x}) + j_{r+1}(\underline{x})$.

□

Corollary 12. *Assume that $\underline{x}$ represents an odd number and let r be the maximal value with $l_r(\underline{x}) < j_0(\underline{x})$. If the algorithm successfully computes $j_i(\underline{x})$ for all i with $l_i(\underline{x}) < j_0(\underline{x})$, then $\underline{x}[j_r(\underline{x}) + l_r(\underline{x}), 0]$ can be replaced by*

$$\tilde{\underline{x}}_r = \sum_{i=0}^{r} (2^{j_i(\underline{x})} - 1) 2^{l_i(\underline{x})}.$$

4 The Algorithm

The algorithm is defined for any n-bit binary digit e and for any $k \geq 2$. We will use following notations.

- $\underline{e}$ is a binary string of length n;
- i is pointer indicating the current position in $\underline{e}$;
- `weight[-1...n - 1]` is an array of integers initialized to n.

In our algorithm, we first look for the start of 1-runs, i.e. the least significant position of such a run (Fig. 2). Once we have found the start of a 1-run, we examine all possible choices of j_0 (Fig. 3). The subroutine `CORE` has as input the starting point i of a substring and a position j_0 which terminates a 1-run. It executes the algorithm given in Fig. 1. If it succeeds in computing all j_i's and l_i's, then it signals `success = TRUE` and returns the weight N_{j_0} and a pointer `top` to the most significant position which is affected by this computation.
As an example, we compute the weight of a 4-SR representations of the string 1 1111 0101.

i	n	n	n	n	n	n	n	n
0	n	n	n	n	2	n	n	1
1	n	n	n	n	2	n	1	1
2	n	n	n	n	2	n	2	1	1
3	n	n	n	n	2	2	2	1	1
4	n	3	n	n	2	2	2	1	1
5	3	3	n	n	2	2	2	1	1

```
i:= 0;
weight[-1]:= 0;
WHILE i < n do
      IF e_i = 0 THEN
            weight[i]:= MIN(weight[i],weight[i − 1]);
            i:= i + 1;
        END THEN
        ELSE REPLACE(i)
      END WHILE
```

Fig. 2. The main algorithm

```
REPLACE(i):
j_0 := position of the next 0 above position i;
IF j_0 > i + k
  THEN weight[i + k]:= MIN(weight[i + k],weight[i − 1]+1);
       i:= i + k;
  END THEN
  ELSE weight[j_0]:= MIN(weight[j_0],weight[i − 1]+1);
       WHILE j_0 ≤ i + k DO
            j_0:= terminator of the next 1-run;
            CORE(i,j_0,top,N_{j_0},success);
            IF  success=TRUE THEN
                weight[top]:= MIN(weight[top],weight[i − 1]+N_{j_0})
       END WHILE
       i:= i+1
  END ELSE
RETURN
```

Fig. 3. Subroutine REPLACE

Using a backtracking algorithm, we would obtain the 4-SR representation 0 00F0 0077. For our previous example with $k = 15$ and $\underline{x} =$ 11 0111 0101 0001 0010 0101 0101 we get

i	1	1	0	1	1	1	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	1
0	8				6				4								3						2			1
2	8	8			6	6			4			4					3			3			2		2	1	1
4	8	8			8	6	6		6	4			4				3			3			2	2	2	1	1
6	8	8			8	6	6		6	4			4				3				3	2	2	2	2	1	1
9	8	8			8	6	6		6	4			4					3	3	3	3	2	2	2	2	1	1
12	8	8			8	6	6		6	4				4	3	3	3	3	3	3	2	2	2	2	2	1	1
16	7	8			8	6	6		6		4	4	4	4	3	3	3	3	3	3	2	2	2	2	2	1	1
18	7	6			8	6	6		5	4	4	4	4	4	3	3	3	3	3	3	2	2	2	2	2	1	1
20	7	6			6	6	6	5	5	4	4	4	4	4	3	3	3	3	3	3	2	2	2	2	2	1	1

5 Analysis of the Algorithm

Define the cost of the algorithm to be the number of read operations on sequence elements.

Theorem 13. *The cost of our algorithm is $O(nk^2)$.*

Proof: The main algorithm reads every position exactly once. For each position i, we have to consider less than k choices for j_0. For each choice, we have to correct less than k positions checking less than $2k$ positions overall. □

Remark: For $k = 6$ and $\underline{x} = 1\ 0110\ 0101$ our algorithm only finds 6-SR representations of weight 4, despite the existence of a 6-SR representation of weight 3, $70\ 00(2^6 - 1)7$. We can change our algorithm so that it chooses Strategy 2 whenever possible. Note that this may force us to check the entire binary representation of e for every choice of j_0. We get

Theorem 14. *The cost of a Strategy 2–algorithm is $O(n^2k^2)$.*

To see the limitations of this approach, consider the number 1101101011. Here, the modified algorithm would not terminate and we have to add further rules defining when to switch from Strategy 2 to Strategy 1.

Open question: Find an indicator to determine which strategy to choose. An 'increase of weights in a 1-run' is a candidate, e.g.

1	0	1	1	0	0	1	0	1
4	3	$\underline{3}$	$\underline{2}$	2	2	2	1	1

Finally, we may decide to pursue both strategies for every corrector. This is guaranteed to give a minimal representation but has exponential complexity.

6 Conclusion

We have proposed a standard representation for k-SR numbers and have shown that all integers have a minimal k-SR representation of this form. Furthermore, we have derived properties which will hold for minimal representations and proposed an algorithm based on these properties. The minimal representation can save more modular multiplications in the 'square and multiply' algorithm than signed-digit representations or canonical k-SR representations [1, 2].

References

1. D. Gollmann, Yongfei Han, and C. Mitchell. Redundant integer representations and fast exponentiation. *to appear in International Journal: Designs, Codes and Cryptography.*
2. D.E. Knuth. *The art of computer programming, Volume 2: seminumerical algorithms.* Addison-Wesley, Reading, Mass., 2nd edition, 1981.
3. R.L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, **21**:120–126, 1978.