Groups Complex. Cryptol. **?** (**????**), 1–16
DOI 10.1515/GCC.****.***

A New Generic Digital Signature Algorithm

Jennifer Seberry, Vinhbuu To and Dongvu Tonien

Abstract. In this paper, we study two digital signature algorithms, the DSA and ECDSA, which have became NIST standard and have been widely used in almost all commercial applications. We will show that the two algorithms are actually 'the same' algebraically and propose a generic algorithm such that both DSA and ECDSA are instances of it.

By looking at this special angle through the generic algorithm, we gain a new insight into the two algorithms DSA and ECDSA. Our new proposed digital signature algorithm is described generically using a group G and a map $\texttt{toNumber} : G \to \mathbb{Z}$. As an illustration, we choose G to be a group of non-singular circulant matrices over finite field and describe a totally new concrete digital signature algorithm.

Keywords. digital signature, discrete log problem, circulant matrices group.

2010 Mathematics Subject Classification. 94A60, 68Q17, 12Y05.

1 Introduction

A digital signature algorithm is a public key cryptographic algorithm designed to protect the authenticity of a digital message or document. A message is signed by a secret key to produce a signature and the signature is verified against the message by a public key. Thus any party can verify the signatures but only one party with the secret key can sign the messages. A valid digital signature gives a recipient reason to believe that the message was created by a known sender who possesses the secret key, and that it was not altered in transit.

Digital signatures are used widely in e-commerce applications, in banking applications, in software distribution, and in other cases where jurisdiction is involved and it is important to detect forgery or tampering. Thus it is crucial to use algorithms that have been standardized by government organizations. Even though there are a numerous number of digital signature algorithms in research literature, only three algorithms have been standardized by the National Institute of Standards and Technology (NIST) and have been widely used in almost all commercial applications. These are the RSA, the DSA and the ECDSA [3, 5].

The RSA was invented by Rivest, Shamir and Adleman, and the security of the algorithm is based on the hardness of factoring a product of two large prime numbers. The DSA was proposed by NIST and attributed to Kravitz, a former

NSA employee. The security of the DSA is based on the hardness of the discrete log problem on the multiplicative group of units on the finite field F_p. The ECDSA is the elliptic curve analogous of the DSA and its security is based on the discrete log problem on the group of points on elliptic curve over a finite field.

In this paper, we study the DSA and ECDSA. Our goal is to find a common algebraic structure between the two algorithms. We show that the two algorithms are actually 'the same' algebraically. We will propose a generic algorithm that captures the common algebraic structure of these two algorithms.

DSA and ECDSA have been standardized and widely used in real world applications. Their securities have been attested by cryptographic community for almost two decades. It is reasonable to believe that our new DSA-based generic algorithm is secure. We will give a proof of the correctness of the generic algorithm and show that the security of the algorithm is based on the hardness of the discrete log problem in the underlined group.

The rest of the paper is organised as follows. In section 2, we give some mathematical preliminaries and notations. In section 3, we compare the two algorithms DSA and ECDSA and derive their algebraic similarity. The new generic algorithm will be described in section 4 and an example is given in section 6. In this example, we will use a group of circulant matrices and describe a concrete digital signature algorithm.

2 Preliminaries

2.1 Groups

We recall that a group is a special algebraic structure that consists of a set $\mathcal{G}$ with a binary operation $* : \mathcal{G} \times \mathcal{G} \to \mathcal{G}$, $(a, b) \mapsto a * b$, that satisfies three conditions: (1) there is a special element e called the *identity* such that $a * e = e * a = a$ for any group element a, (2) for each group element a, there is a group element a^{-1}, called the inverse of a, such that $a * a^{-1} = a^{-1} * a = e$, and (3) the operation is associative, that is $a * (b * c) = (a * b) * c$ for any group elements a, b and c.

For an integer n and a group element a, we define the power a^n as $a^n = a * a * \cdots * a$ (n times). If the group is commutative, people often use $+$ for the group operation and 0 for the identity element. In this additive notation, the power function is $a+a+\cdots+a = na$. In this paper, we will discuss algebraic properties generically, we are going to prototype group operations as in modern programming language as follows

```
GroupElement multiply(GroupElement a, GroupElement b);
GroupElement inverse(GroupElement a);
GroupElement pow(GroupElement a, Integer n);
```

The correctness proofs of the DSA, ECDSA and our new algorithm are based on the following classical theorem.

Theorem 2.1. (Lagrange's Theorem) *In a finite group $\mathcal{G}$, for any group element a,* $\texttt{pow}(\texttt{a}, |\mathcal{G}|)$ *is equal to the identity element.*

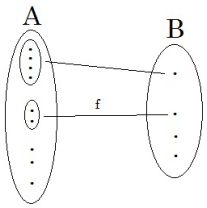
The *discrete log problem* in a group $\mathcal{G}$ is stated as follows: given two group elements a and $b = \texttt{pow}(a, x)$, find the number x. In DSA and ECDSA, we need to use groups such that the discrete log problem is hard. This is because in these algorithms, the attacker knows the two group elements g and y. The element g is the group generator and is specified in the group parameters. The element y is the public key. Both g and y are made public and anyone will know these values. The secret key is the number x and we have $y = \texttt{pow}(g, x)$. If the discrete log problem is not hard then from g and y, anyone can solve the discrete log problem to obtain the secret key x.

2.2 Fiber sets and fiber-skew functions

For a function $f : A \to B$ and $y \in B$, the fiber set F_y is defined as follows

$$F_y = \{x \in A : f(x) = y\}.$$

F_y is the set of all elements of A that are mapped to y in B.



The domain A is partitioned into the fiber sets F_y. If A and B are finite sets then the size of the fiber set F_y reflects the probability $Pr[x : f(x) = y]$. Indeed,

$$Pr[x : f(x) = y] = \frac{|F_y|}{|A|}.$$

If $f : A \to B$ is a random function then each $y \in B$ will have an equal probability for $f(x) = y$. It means that all the fiber sets F_y have approximately the same size and $|F_y| \approx |A|/|B|$. On the other hand, if there is a fiber set F_y that has a relatively larger size then with this particular value of y, there is a larger

probability that $f(x) = y$. In this case, we call the function f as a *skew-fiber function*.

To measure how skew a function is, we may use the quantity

$$\alpha = \frac{\max(|F_y|)}{|A|} = \max_{y \in B}(Pr[x : \ f(x) = y]).$$

then $\alpha \in [1/|B|, 1]$ and the larger the α value the more skew a function is. In this sense, a random function is a least skew function with $\alpha = 1/|B|$. A constant function is a most skew function with $\alpha = 1$. In a constant function, one fiber set is equal to the whole set B whereas other fiber sets are empty.

We will see later that in our new generic algorithm, one of our security requirements is that the function $\texttt{toNumber}_n : \mathcal{G} \to \mathbb{Z}_n$ is not a skew-fiber function. This requirement means that for each $i \in \mathbb{Z}_n$, the probability that $\texttt{toNumber}_n(g) = i$ is negligible, or equivalently, all the fiber sets have roughly similar size and no fiber set has a larger size.

3 Comparison between DSA and ECDSA

In this section, we will compare the two algorithms DSA and ECDSA. We will look at DSA and ECDSA step by step in details.

Domain parameters setup.

DSA Setup	*ECDSA Setup*
p is a prime, q is a prime divisor of $p-1$	$E(F_q)$ is an elliptic curve over the field F_q
g is an element of order q in $(F_p^*, \times)$	G is an element of prime order n in $(E(F_q), +)$
Domain parameters: p, q, g	Domain parameters: $E(F_q)$, n, G

We can see that in the setup phase, the aim in both DSA and ECDSA is to set up a group $(\mathcal{G}, *)$ such that the discrete log problem is hard.

In DSA, $\mathcal{G}$ is the subgroup of $(F_p^*, \times)$ generated by the element g of order q:

$$\mathcal{G} = \langle g \rangle = \{1, g, g^2, \ldots, g^{q-1}\},$$

and the group operation $*$ is the multiplication modulo p.

In ECDSA, $\mathcal{G}$ is the subgroup of $(E(F_q), +)$ generated by the element G of order n:

$$\mathcal{G} = \langle G \rangle = \{\mathcal{O}, G, 2G, \ldots, (n-1)G\},$$

and the group operation $*$ is the elliptic curve point addition $+$.

Let us use the following prototyped function in programming language to denote the function that returns the order of the group $\mathcal{G}$.

`Integer getOrder();`

In DSA, `getOrder()` $= q$, and in ECDSA, `getOrder()` $= n$.

Key generation.

DSA Key generation	*ECDSA Key generation*
Select a random integer $x \in [1, q-1]$	Select a random integer $d \in [1, n-1]$
Compute $y = g^x \pmod p$	Compute $Q = dG$
The private key is x	The private key is d
The public key is y	The public key is Q

The key generation of both DSA and ECDSA can be re-written generically as follows

Select a random integer $x \in [1, \texttt{getOrder}() - 1]$.
Compute $y = \texttt{pow}(\texttt{g}, \texttt{x})$.
The private key is $x \in \mathbb{Z}$.
The public key is $y \in \mathcal{G}$.

Signature generation and verification

DSA Signature generation	*ECDSA Signature generation*
1. Compute $e = H(M)$	1. Compute $e = H(M)$
2. Select a random integer $k \in [1, q-1]$	2. Select a random integer $k \in [1, n-1]$
3. Compute $w = g^k \pmod p$	3. Compute $W = kG = (x_1, y_1)$
4. Compute $r = w \pmod q$	4. Compute $r = x_1 \pmod n$
5. Compute $s = k^{-1}(e + xr) \pmod q$	5. Compute $s = k^{-1}(e + dr) \pmod n$
6. The signature for M is (r, s)	6. The signature for M is (r, s)

DSA Signature verification	*ECDSA Signature verification*
1. Compute $e = H(M)$	1. Compute $e = H(M)$
2. Compute $z = s^{-1} \pmod q$	2. Compute $z = s^{-1} \pmod n$
3. Compute $u_1 = ez \pmod q$	3. Compute $u_1 = ez \pmod n$
4. Compute $u_2 = rz \pmod q$	4. Compute $u_2 = rz \pmod n$
5. Compute $v = g^{u_1} y^{u_2} \pmod p$	5. Compute $V = u_1 G + u_2 Q = (x_1, y_1)$
6. Compute $r' = v \pmod q$	6. Compute $r' = x_1 \pmod n$
7. Accept the signature if $r' = r$	7. Accept the signature if $r' = r$

We can see that DSA and ECDSA are different in the signature generation and verification. In signature generation, the difference is in the step 4 and in signature verification, the difference is in the step 6.

In order to resolve these differences, we propose to rewrite the step 4 in signature generation into two steps 4A and 4B. Also in signature verification, we will rewrite the step 6 into two steps 6A and 6B as follows.

Rewriting signature generation and verification

DSA Signature generation	*ECDSA Signature generation*
1. Compute $e = H(M)$	1. Compute $e = H(M)$
2. Select a random integer $k \in [1, q-1]$	2. Select a random integer $k \in [1, n-1]$
3. Compute $w = g^k \pmod p$	3. Compute $W = kG = (x_1, y_1)$
4A. Compute $t = w$	4A. Compute $t = x_1$
4B. Compute $r = t \pmod q$	4B. Compute $r = t \pmod n$
5. Compute $s = k^{-1}(e + xr) \pmod q$	5. Compute $s = k^{-1}(e + dr) \pmod n$
6. The signature for M is (r, s)	6. The signature for M is (r, s)
DSA Signature verification	*ECDSA Signature verification*
1. Compute $e = H(M)$	1. Compute $e = H(M)$
2. Compute $z = s^{-1} \pmod q$	2. Compute $z = s^{-1} \pmod n$
3. Compute $u_1 = ez \pmod q$	3. Compute $u_1 = ez \pmod n$
4. Compute $u_2 = rz \pmod q$	4. Compute $u_2 = rz \pmod n$
5. Compute $v = g^{u_1} y^{u_2} \pmod p$	5. Compute $V = u_1 G + u_2 Q = (x_1, y_1)$
6A. Compute $t = v$	6A. Compute $t = x_1$
6B. Compute $r' = t \pmod q$	6B. Compute $r' = t \pmod n$
7. Accept the signature if $r' = r$	7. Accept the signature if $r' = r$

We can see that the step 4A of the DSA signature generation and the step 6A of the DSA signature verification seem redundant. However, this is the crucial step of our proposed generic algorithm. An important fact that we want to point out is that in step 4A of DSA, even t and w have the same value but they are of different types. The variable w is a group element of $\mathcal{G}$, but the variable t is an integer. Therefore, step 4A seems redundant but it is not.

To make this point clear, we will introduce the following function

```
Integer  toNumber(GroupElement  a);
```

The function `toNumber` : $\mathcal{G} \to \mathbb{Z}$ is a function maps a group element to an integer.

In DSA, the function `toNumber` : $\mathcal{G} \to \mathbb{Z}$ is the trivial identity function that map each group element $a \in F_p^*$ to the number $a \in [1, p-1]$

```
Integer  toNumber(GroupElement  a){
return a;
}
```

In ECDSA, the function `toNumber` : $\mathcal{G} \to \mathbb{Z}$ is the function that map each group element $a \in E(F_q)$ to the x-coordinate of a

```
Integer  toNumber(GroupElement  a){
write the point as a = (x_1, y_1)
return x_1;
}
```

With the introduction of the function `toNumber` : $\mathcal{G} \to \mathbb{Z}$, we can see that the two algorithms DSA and ECDSA are now exactly the same "algebraically".

4 Generic algorithm

In this section, we will describe our new generic digital signature algorithm which is motivated from the two algorithm DSA and ECDSA. This generic algorithm will use a group $(\mathcal{G}, *)$ such that the discrete log problem is hard.

We will use a function `toNumber` : $\mathcal{G} \to \mathbb{Z}$ that maps a group element $a \in \mathcal{G}$ to an integer:

```
Integer  toNumber(GroupElement  a);
```

We will later show that our generic algorithm is correct and its correctness is independent of the choice of the group $\mathcal{G}$ and the function `toNumber` : $\mathcal{G} \to \mathbb{Z}$.

Below is the description of the generic algorithm

Setup.

$(\mathcal{G}, *)$ is a cyclic group
g is a generator of $(\mathcal{G}, *)$
n is the order of the group $(\mathcal{G}, *)$
Domain parameters: description of $(\mathcal{G}, *)$, g, n

Key generation.

Select a random integer $x \in [1, n-1]$
Compute $y = \mathtt{pow}(g, x)$
The private key is $x \in \mathbb{Z}$

The public key is $y \in \mathcal{G}$

Signature generation.

1. Compute $e = H(M)$
2. Select a random integer $k \in [1, n-1]$
3. Compute $w = \texttt{pow}(g, k) \in \mathcal{G}$
4A. Compute $t = \texttt{toNumber}(w) \in \mathbb{Z}$
4B. Compute $r = t \pmod{n}$
5. Compute $s = k^{-1}(e + xr) \pmod{n}$
6. The signature for M is (r, s)

Signature verification.

1. Compute $e = H(M)$
2. Compute $z = s^{-1} \pmod{n}$
3. Compute $u_1 = ez \pmod{n}$
4. Compute $u_2 = rz \pmod{n}$
5. Compute $v = \texttt{multiply}(\texttt{pow}(g, u_1), \texttt{pow}(y, u_2)) \in \mathcal{G}$
6A. Compute $t = \texttt{toNumber}(v) \in \mathbb{Z}$
6B. Compute $r' = t \pmod{n}$
7. Accept the signature if $r' = r$

5 Security analysis of the generic algorithm

5.1 Proof of correctness

We now prove the correctness of the generic algorithm. We show that if (r, s) is a signature of a message M generated by the signature generation algorithm then the signature verification algorithm must accept this signature as valid. This correctness of the algorithm is independent of the choice of the group $\mathcal{G}$ and the function $\texttt{toNumber} : \mathcal{G} \to \mathbb{Z}$.

Theorem 5.1. *If (r, s) is a signature generated by the signature generation algorithm of a message M then there must exist a number k such that*

$$\begin{aligned} r &= \texttt{toNumber}(\texttt{pow}(g, k)) \pmod{n} \\ s &= k^{-1}(e + xr) \pmod{n}, \end{aligned}$$

where $e = H(M) \in \mathbb{Z}$.

Theorem 5.2. *A valid signature (r, s) that passes the signature verification algorithm for a message M must satisfy*

$$r \;=\; \texttt{toNumber}(\texttt{pow}(g, z(e + xr)) \pmod{n}$$

where $e = H(M) \in \mathbb{Z}$ and $z = s^{-1} \pmod{n}$.

Proof. From the signature verification algorithm, we have

$$\begin{aligned} r &= t \pmod{n} = \texttt{toNumber}(v) \pmod{n} \\ &= \texttt{toNumber}(\texttt{pow}(g, u_1), \texttt{pow}(y, u_2)) \pmod{n} \\ &= \texttt{toNumber}(\texttt{pow}(g, u_1 + xu_2)) \pmod{n} \\ &= \texttt{toNumber}(\texttt{pow}(g, z(e + xr))) \pmod{n}. \quad \diamondsuit \end{aligned}$$

The correctness of the digital signature algorithm now follows from Theorem 5.1 and Theorem 5.2. Indeed, by Theorem 5.1, if r and s are generated by the signature generation algorithm then $s = k^{-1}(e + xr) \pmod{n}$. Therefore, $z(e + xr) = k \pmod{n}$, so $k = z(e + xr) + nj$ for some $j \in \mathbb{Z}$. Thus,

$$\begin{aligned} r &= \texttt{toNumber}(\texttt{pow}(g, k)) \pmod{n} \\ &= \texttt{toNumber}(\texttt{pow}(g, z(e + xr) + nj)) \pmod{n} \\ &= \texttt{toNumber}(\texttt{pow}(g, z(e + xr))) \pmod{n}. \end{aligned}$$

The last equality is derived from the Lagrange's theorem which asserts that for any element g of a finite group $\mathcal{G}$ of order n, $\texttt{pow}(g, n)$ is the identity element of the group. This shows that (r, s) is a valid signature for M and the generic algorithm is correct.

5.2 Secretcy of the private key

The generator g of the group $\mathcal{G}$ is specified in the group parameters. The element y is the public key. So both g and y are made public and anyone knows these values. The secret key is the number x and we have $y = \texttt{pow}(g, x)$. Therefore, in order to keep x secret, the discrete log problem in $\mathcal{G}$ must be hard. If the discrete log problem is not hard then from g and y, anyone can solve the discrete log problem to obtain the secret key x and subsequently can forge a valid signature of any message.

5.3 Protection against forgery attack

In order for an attacker to forge a valid signature (r, s) that passes the signature verification algorithm, the attacker must generate M, r and s that satisfy the following condition

$$r \;=\; \texttt{toNumber}(\texttt{pow}(g, z(e + xr)) \pmod n$$

where $e = H(M) \in \mathbb{Z}$ and $z = s^{-1} \pmod n$.

Since n is a public information, the ability to generate s is equivalent to the ability to generate z. Therefore, in order to forge a valid signature, the attacker must be able to generate M, r and z that satisfy the following condition

$$r \;=\; \texttt{toNumber}((g^{H(M)}y^r)^z) \pmod n. \tag{1}$$

Consider the function toNumber modulo n, which we will denoted by $\texttt{toNumber}_n$

$$\begin{aligned} \texttt{toNumber}_n : G &\to \mathbb{Z}_n \\ a &\mapsto \texttt{toNumber}(a) \pmod n \end{aligned}$$

For each $i \in \mathbb{Z}_n$, let F_i denote the fiber set of i, that is

$$F_i = \{a \in \mathcal{G} : \texttt{toNumber}_n(a) = i\}.$$

Then the condition (1) is equivalent to $(g^{H(M)}y^r)^z \in F_r$.

Theorem 5.3. *If an attacker can forge a valid signature then it can generate M, r and z such that*

$$(g^{H(M)}y^r)^z \in F_r. \tag{2}$$

The following theorem states four requirements for the group $\mathcal{G}$, the function $\texttt{toNumber} : \mathcal{G} \to \mathbb{Z}$ and the hash function H so that the generic algorithm is secure against forgery attack. Note that our first three requirements are similar to the ones stated in [2] but they are a bit stronger because we consider them in $\mathbb{Z}_n$ instead of in $\mathbb{Z}$ as in [2].

Theorem 5.4. *The following conditions are necessary security conditions for the generic algorithm*

(i) *The discrete log problem in the group $\mathcal{G}$ is hard;*

(ii) *The hash function modulo n (i.e. $M \mapsto H(M) \pmod n$) is a one-way function;*

(iii) *The hash function modulo n is a collision-resistant function;*

(iv) *The function* $\texttt{toNumber}_n$ *is not a fiber-skew function.*

Proof. The first condition is the crucial one. If the discrete log problem in the group $\mathcal{G}$ is not hard then the attacker can recover the secret key from the public key and forge signature on any message. Alternatively, for any message M, the attacker can choose a random group element a and calculate $\texttt{toNumber}(a) \pmod n = r$. This gives $a \in F_r$. The attacker then solves the discrete log problem to find z such that $(g^{H(M)}y^r)^z = a$ and so $(g^{H(M)}y^r)^z \in F_r$. By Theorem 5.3, this gives a valid signature on M.

The second condition requires that given a number t, it is hard to find M such that $H(M) \pmod n = t$. If this is not hard then the attacker can formulate a valid signature as follows. The attacker will choose random u and v and try to find M, r, z such that $(g^{H(M)}y^r)^z = g^u y^v \in F_r$. First, the attacker calculates $\texttt{toNumber}(g^u y^v) \pmod n = r$, this gives $g^u y^v \in F_r$. Then the attacker calculates $z = vr^{-1} \pmod n$, this gives $y^v = y^{rz}$. Next, the attacker calculates $t = uz^{-1} \pmod n$, this gives $g^u = g^{tz}$. Finally the attacker calculates M by inverting the hash function modulo n from the equation $H(M) = t \pmod n$ and so $(g^{H(M)}y^r)^z = g^u y^v \in F_r$.

The third condition requires that it is hard to find two different messages M_1 and M_2 such that $H(M_1) = H(M_2) \pmod n$ because if two such messages can be found easily then a valid signature for one message will become a valid signature for the other.

The last condition requires that no fiber set F_i has a large size. If for example, there is a number $i \in \mathbb{Z}_n$ such that the fiber set F_i is large, then the attacker can forge a signature as follows. First the attacker will choose a number $r \in \mathbb{Z}_n$ such that F_r is a large set. Then the attacker choose random M and z, this makes $(g^{H(M)}y^r)^z$ a random element of the group $\mathcal{G}$. Since F_r is a large set, the probability that $(g^{H(M)}y^r)^z$ belongs to the fiber set F_r is not negligible. Therefore, the probability that the attacker can form a valid signature is not negligible. $\diamondsuit$

The condition (4) in Theorem 5.4 is a reasonable condition. In DSA, `toNumber` is the identity function so this condition is satisfied. In ECDSA, the function `toNumber` returns the x-coordinate of the elliptic point so this condition is likely to be satisfied. Since this condition requires that no fiber set F_r have large size, for a random group element a, the probability for $a \in F_r$ is negligible. From the first three conditions, $(g^{H(M)}y^r)^z$ is a random group element so the condition $(g^{H(M)}y^r)^z \in F_r$ in Theorem 5.3 will be satisfied with a negligible probability.

6 DSA based on circulant matrices over finite field

In the previous section, we propose a generic algorithm that defined over a group $\mathcal{G}$ and a function $\texttt{toNumber} : \mathcal{G} \to \mathbb{Z}$. The security requirement is that the discrete log problem in $\mathcal{G}$ is hard. In this chapter, we choose $\mathcal{G}$ to be the group of non-singular circulant matrices over a finite field. This group is recently studied by Mahalanobis [1] and it is believed that in this group the discrete log problem is hard. With this specific choice of the underlined group, the generic algorithm gives us a totally new concrete digital signature algorithm.

6.1 Circulant matrices

The group of non-singular circulant matrices over finite field is recently studied by Mahalanobis [1]. A $d \times d$ matrix over a field F is called *circulant*, if every row except the first row, is a right circular shift of the row above that. An example of a circulant 5×5 matrix is:

$$\begin{pmatrix} c_0 & c_1 & c_2 & c_3 & c_4 \\ c_4 & c_0 & c_1 & c_2 & c_3 \\ c_3 & c_4 & c_0 & c_1 & c_2 \\ c_2 & c_3 & c_4 & c_0 & c_1 \\ c_1 & c_2 & c_3 & c_4 & c_0 \end{pmatrix}$$

So a circulant matrix is defined by its first row. We will denote a circulant matrix C with first row $c_0, c_1, \ldots, c_{d-1}$ by $C = \texttt{circ}(c_0, c_1, \ldots, c_{d-1})$. The row-sum of the matrix C is defined as

$$\texttt{rowsum}(C) = c_0 + c_1 + \cdots + c_{d-1}.$$

Let $W = \texttt{circ}(0, 1, 0, \ldots, 0)$ be a $d \times d$ circulant matrix then clearly $W^d = I$ – the identity matrix – and any circulant matrix $C = \texttt{circ}(c_0, c_1, \ldots, c_{d-1})$ can be written as a linear sum of W^i as follows

$$C = c_0 I + c_1 W + c_2 W^2 + \cdots + c_{d-1} W^{d-1}.$$

Since $W^d = I$, the set $\mathcal{C}$ of all circulant matrices with matrix addition and multiplication is isomorphic to the ring of all polynomials $F[x]$ modulo $x^d - 1$. The isomorphism is

$$\begin{aligned} \imath : \mathcal{C} &\to F[x]/(x^d - 1) \\ C = \texttt{circ}(c_0, c_1, \ldots, c_{d-1}) &\mapsto c_0 + c_1 x + c_2 x^2 + \cdots + c_{d-1} x^{d-1}. \end{aligned}$$

Example: Consider 5×5 circulant matrices over the field F_{17}. Let $A = \texttt{circ}(2,7,3,4,2)$ and $B = \texttt{circ}(5,1,6,4,1)$. The corresponding polynomials are

$$\begin{aligned} \imath(A) &= 2 + 7x + 3x^2 + 4x^3 + 2x^4 \\ \imath(B) &= 5 + x + 6x^2 + 4x^3 + x^4. \end{aligned}$$

We have

$$\imath(A) + \imath(B) = 7 + 8x + 9x^2 + 8x^3 + 3x^4,$$

and

$$\begin{aligned} \imath(A) \times \imath(B) &= 10 + 37x + 34x^2 + 73x^3 + 62x^4 + 45x^5 + 31x^6 + 12x^7 + 2x^8 \\ &= 10 + 37x + 34x^2 + 73x^3 + 62x^4 + 45 + 31x + 12x^2 + 2x^3 \\ &= 55 + 68x + 46x^2 + 75x^3 + 62x^4 \\ &= 4 + 12x^2 + 7x^3 + 11x^4. \end{aligned}$$

We will verify that

$$A + B = \texttt{circ}(7,8,9,8,3),$$

and

$$A \times B = \texttt{circ}(4,0,12,7,11).$$

Indeed,

$$A+B = \begin{pmatrix} 2 & 7 & 3 & 4 & 2 \\ 2 & 2 & 7 & 3 & 4 \\ 4 & 2 & 2 & 7 & 3 \\ 3 & 4 & 2 & 2 & 7 \\ 7 & 3 & 4 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 5 & 1 & 6 & 4 & 1 \\ 1 & 5 & 1 & 6 & 4 \\ 4 & 1 & 5 & 1 & 6 \\ 6 & 4 & 1 & 5 & 1 \\ 1 & 6 & 4 & 1 & 5 \end{pmatrix} = \begin{pmatrix} 7 & 8 & 9 & 8 & 3 \\ 3 & 7 & 8 & 9 & 8 \\ 8 & 3 & 7 & 8 & 9 \\ 9 & 8 & 3 & 7 & 8 \\ 8 & 9 & 8 & 3 & 7 \end{pmatrix},$$

and

$$A \times B = \begin{pmatrix} 2 & 7 & 3 & 4 & 2 \\ 2 & 2 & 7 & 3 & 4 \\ 4 & 2 & 2 & 7 & 3 \\ 3 & 4 & 2 & 2 & 7 \\ 7 & 3 & 4 & 2 & 2 \end{pmatrix} \times \begin{pmatrix} 5 & 1 & 6 & 4 & 1 \\ 1 & 5 & 1 & 6 & 4 \\ 4 & 1 & 5 & 1 & 6 \\ 6 & 4 & 1 & 5 & 1 \\ 1 & 6 & 4 & 1 & 5 \end{pmatrix}$$

$$= \begin{pmatrix} 55 & 68 & 46 & 75 & 62 \\ 62 & 55 & 68 & 46 & 75 \\ 75 & 62 & 55 & 68 & 46 \\ 46 & 75 & 62 & 55 & 68 \\ 68 & 46 & 75 & 62 & 55 \end{pmatrix} = \begin{pmatrix} 4 & 0 & 12 & 7 & 11 \\ 11 & 4 & 0 & 12 & 7 \\ 7 & 11 & 4 & 0 & 12 \\ 12 & 7 & 11 & 4 & 0 \\ 0 & 12 & 7 & 11 & 4 \end{pmatrix}.$$

6.2 Discrete log problem for circular matrices

The discrete log problem for circular matrices is stated as follows: given a circular matrix A of size $d \times d$ over the field F_q and a circular matrix $B = A^x$, find the exponent x.

Mahalanobis [1] showed that the discrete log problem for circular matrices is as hard as the discrete log problem in the finite field $F_{q^{d-1}}$ if the matrix A is chosen to satisfy the following five conditions

- Determinant of A is equal to 1,
- $\texttt{rowsum}(A) = 1$,
- The polynomial $\frac{\chi_A}{x-1}$ is irreducible where χ_A is the characteristic polynomial of A,
- d is a prime number,
- q is primitive modulo d.

6.3 DSA based on group of circulant matrices

In this section, we will apply the generic algorithm for a group of circulant matrices. What we obtain is a totally new digital signature algorithm. The underlined group $\mathcal{G}$ is the multiplicative group generated by a circulant matrix A defined over a field F_q that satisfies a number of conditions described in section 6.2.

Suppose that n is the multiplicative order of the matrix A, that is $A^n = I$, then the group $\mathcal{G}$ is

$$\mathcal{G} = \{I, A, A^2, \ldots, A^{n-1}\}.$$

Note that matrices in $\mathcal{G}$ are all circulant matrices.

We will chose an encoding function that maps each field element of F_q to an integer

$$\texttt{encode} : F_q \to \mathbb{Z}.$$

For example, when $q = p$ is a prime number then $\texttt{encode}(a) = a \pmod p$. When $q = p^m$ then each field element a of F_{p^m} is a polynomial over F_p of degree $< m$,

that is, $a = a_0 + a_1x + \cdots + a_{m-1}x^{m-1}$, where $a_0, a_1, \ldots, a_{m-1} \in F_p$. In this case, we can define $\texttt{encode}(a)$ to be the mN-bit number $a_0||a_1||\ldots||a_{m-1}$ where N is the bit length of p. Here each a_i is represented as a N-bit number.

We need to define a function $\texttt{toNumber} : \mathcal{G} \to \mathbb{Z}$ to use in the generic algorithm. We will use the encoding function to define the function `toNumber`. There are many choices for this function `toNumber`. To illustrate, let define the function `toNumber` as follows

$$\begin{aligned} \texttt{toNumber} : \mathcal{G} &\to \mathbb{Z} \\ C = \texttt{circ}(\texttt{c}_0, \texttt{c}_1, \ldots, \texttt{c}_{\texttt{d}-1}) &\mapsto \texttt{encode}(\texttt{c}_0 + \texttt{c}_2 + \texttt{c}_4 + \cdots + \texttt{c}_{\texttt{d}-1}). \end{aligned}$$

Note that the above definition makes sense because $c_0 + c_2 + c_4 + \cdots + c_{d-1}$ is a field element of F_q and so $\texttt{encode}(c_0 + c_2 + c_4 + \cdots + c_{d-1})$ is an integer.

Below is the description of the DSA algorithm based on circulant matrices.

Set up.

Choose a circulant matrix $A = \texttt{circ}(a_0, a_1, \ldots, a_{d-1})$ of size $d \times d$ over F_q
n is the multiplicative order of A
Domain parameters: description of $a_0, a_1, \ldots, a_{d-1}, n$

Key generation.

Select a random integer $x \in [1, n-1]$
Compute $B = A^x = \texttt{circ}(b_0, b_1, \ldots, b_{d-1})$
The private key is $x \in \mathbb{Z}$
The public key is $b_0, b_1, \ldots, b_{d-1}$

Signature generation.

Compute $e = H(M)$
Select a random integer $k \in [1, n-1]$
Compute $C = A^k = \texttt{circ}(c_0, c_1, \ldots, c_{d-1})$
Compute $t = \texttt{encode}(c_0 + c_2 + c_4 + \cdots + c_{d-1}) \in \mathbb{Z}$
Compute $r = t \pmod n$
Compute $s = k^{-1}(e + xr) \pmod n$
The signature for M is (r, s)

Signature verification.

Compute $e = H(M)$
Compute $z = s^{-1} \pmod n$

Compute $u_1 = ez \pmod{n}$
Compute $u_2 = rz \pmod{n}$
Compute $V = A^{u_1}B^{u_2} = \texttt{circ}(v_0, v_1, \ldots, v_{d-1})$
Compute $t = \texttt{encode}(v_0 + v_2 + v_4 + \cdots + v_{d-1}) \in \mathbb{Z}$
Compute $r' = t \pmod{n}$
Accept the signature if $r' = r$

Bibliography

[1] A. Mahalanobis, The discrete logarithm problem in the group of non-singular circulant matrices, *Groups Complexity Cryptology* **2** (2010), 83–89.

[2] S. Vaudenay, The security of DSA and ECDSA, in: *Public Key Cryptography – PKC 2003*, Lecture Notes in Computer Science 2567, Springer, Berlin (2002), 309–323.

[3] National Institute of Standards and Technology, *Digital Signature Standard (DSS)*, Federal Information Processing Standards 186–3.

[4] National Institute of Standards and Technology, *Secure Hash Standard (SHS)*, Federal Information Processing Standards 180–3.

[5] American National Standards Institute, *Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, American National Standards Institute X9.62.

Received **???**.

Author information

Jennifer Seberry, School of Computer Science and Software Engineering, University of Wollongong, Australia.
E-mail: jennie@uow.edu.au

Vinhbuu To, School of Computer Science and Software Engineering, University of Wollongong, Australia.
E-mail: vinhbuu.to@gmail.com

Dongvu Tonien, Mathematical Sciences Institute, Australian National University, Australia.
E-mail: dongvu.tonien@gmail.com