# Digital signature scheme using non-square matrices

**Jiale Chen, Dima Grigoriev, Vladimir Shpilrain**

Computability in Europe, July 2023

# Setup

Let $K = \mathbb{Z}_q[x_1, \ldots, x_n]$ denote the algebra of polynomials in $n$ variables over the ring $\mathbb{Z}_q$ of integers modulo $q$, where an integer $q$ is not necessarily a prime.

# Setup

Let $K = \mathbb{Z}_q[x_1, \ldots, x_n]$ denote the algebra of polynomials in $n$ variables over the ring $\mathbb{Z}_q$ of integers modulo $q$, where an integer $q$ is not necessarily a prime.

**Public:**

– $k \times l$ left invertible matrix $M$, with $k > l$, whose entries are sparse polynomials from the algebra $K$.

– a hash function $H$ (e.g., SHA-512) and a (deterministic) procedure for converting values of $H$ to vectors of sparse polynomials from the algebra $K$.

# Setup

Let $K = \mathbb{Z}_q[x_1, \ldots, x_n]$ denote the algebra of polynomials in $n$ variables over the ring $\mathbb{Z}_q$ of integers modulo $q$, where an integer $q$ is not necessarily a prime.

**Public:**

– $k \times l$ left invertible matrix $M$, with $k > l$, whose entries are sparse polynomials from the algebra $K$.

– a hash function $H$ (e.g., SHA-512) and a (deterministic) procedure for converting values of $H$ to vectors of sparse polynomials from the algebra $K$.

**Private:** $l \times k$ right invertible matrix $L$ over $K$, such that $LM$ is the $l \times l$ identity matrix.

# Signature scheme

**Signing** a message $m$:

1. Apply a hash function $H$ to $m$. Convert $H(m)$ to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of $l$ (sparse) polynomials from the algebra $K$ using a deterministic public procedure.
2. Multiply the vector $\mathbf{U}$ by the (private) matrix $L$ on the right to get a vector $\mathbf{V} = \mathbf{U}L = (Q_1, \ldots, Q_k)$ of $k$ polynomials from $K$.
3. The signature is the vector $\mathbf{V}$.

# Verification

1. The verifier computes the hash $H(m)$ and converts $H(m)$ to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of $l$ (sparse) polynomials using a deterministic public procedure.
2. The verifier multiplies the signature vector $\mathbf{V}$ by the public matrix $M$ on the right to get a vector $\mathbf{W}$.
3. The signature is accepted if and only if $\mathbf{W} = \mathbf{U}$.

# Verification

1. The verifier computes the hash $H(m)$ and converts $H(m)$ to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of $l$ (sparse) polynomials using a deterministic public procedure.
2. The verifier multiplies the signature vector $\mathbf{V}$ by the public matrix $M$ on the right to get a vector $\mathbf{W}$.
3. The signature is accepted if and only if $\mathbf{W} = \mathbf{U}$.

**Correctness** is obvious since $\mathbf{W} = \mathbf{V}M = (\mathbf{U}L)M = \mathbf{U}(LM) = \mathbf{U}$.

# Generating a (left) invertible $k \times l$ matrix

Let $k > l$. To generate a (left) invertible $k \times l$ matrix, one can first generate an invertible square $k \times k$ matrix and then remove $k - l$ columns, selected at random.

# Generating a (left) invertible $k \times l$ matrix

Let $k > l$. To generate a (left) invertible $k \times l$ matrix, one can first generate an invertible square $k \times k$ matrix and then remove $k - l$ columns, selected at random.

To generate an invertible square $k \times k$ matrix, one can do the following.

1. Generate an upper unitriangular $k \times k$ matrix $U$ as a product of *elementary matrices* $E_{ij}(u)$. A matrix $E_{ij}(u)$ has 1s on the diagonal and 0s elsewhere, except that it has a polynomial $u = u(x_1, \ldots, x_n)$ in the $(i, j)$th place, where $j > i$.
2. Thus, for every pair of integers $(i, j)$ with $1 \leq i < j \leq k$, select a random $t$-sparse polynomial $u = u_{ij}$ and make an elementary matrix $E_{ij}(u)$.
3. Finally, an upper unitriangular $k \times k$ matrix $U$ is computed as a product of $\frac{k^2-k}{2}$ elementary matrices selected that way.

# Generating a (left) invertible $k \times l$ matrix

Let $k > l$. To generate a (left) invertible $k \times l$ matrix, one can first generate an invertible square $k \times k$ matrix and then remove $k - l$ columns, selected at random.

To generate an invertible square $k \times k$ matrix, one can do the following.

1. Generate an upper unitriangular $k \times k$ matrix $U$ as a product of *elementary matrices* $E_{ij}(u)$. A matrix $E_{ij}(u)$ has 1s on the diagonal and 0s elsewhere, except that it has a polynomial $u = u(x_1, \ldots, x_n)$ in the $(i, j)$th place, where $j > i$.
2. Thus, for every pair of integers $(i, j)$ with $1 \le i < j \le k$, select a random $t$-sparse polynomial $u = u_{ij}$ and make an elementary matrix $E_{ij}(u)$.
3. Finally, an upper unitriangular $k \times k$ matrix $U$ is computed as a product of $\frac{k^2-k}{2}$ elementary matrices selected that way.
4. A lower unitriangular $k \times k$ matrix $K$ is built a similar way, except that in the elementary matrices $E_{ij}(u)$, one should have $1 \le j < i \le k$.

# Generating a (left) invertible $k \times l$ matrix

Let $k > l$. To generate a (left) invertible $k \times l$ matrix, one can first generate an invertible square $k \times k$ matrix and then remove $k - l$ columns, selected at random.

To generate an invertible square $k \times k$ matrix, one can do the following.

1. Generate an upper unitriangular $k \times k$ matrix $U$ as a product of *elementary matrices* $E_{ij}(u)$. A matrix $E_{ij}(u)$ has 1s on the diagonal and 0s elsewhere, except that it has a polynomial $u = u(x_1, \ldots, x_n)$ in the $(i, j)$th place, where $j > i$.
2. Thus, for every pair of integers $(i, j)$ with $1 \leq i < j \leq k$, select a random $t$-sparse polynomial $u = u_{ij}$ and make an elementary matrix $E_{ij}(u)$.
3. Finally, an upper unitriangular $k \times k$ matrix $U$ is computed as a product of $\frac{k^2-k}{2}$ elementary matrices selected that way.
4. A lower unitriangular $k \times k$ matrix $K$ is built a similar way, except that in the elementary matrices $E_{ij}(u)$, one should have $1 \leq j < i \leq k$.
5. An invertible $k \times k$ square matrix $S$ is now computed as a product $UP_1KP_2$, where $P_i$ are matrices corresponding to random permutations of columns and rows of a $k \times k$ matrix. (Entries of $P_i$ are 0s and 1s.)

# Computing the (left) inverse of a matrix

Having generated an invertible $k \times k$ square matrix $S = UP_1KP_2$, we compute its inverse as $S^{-1} = P_2^{-1}K^{-1}P_1^{-1}U^{-1}$. Computing $P_i^{-1}$ is trivial, and computing the inverse of a unitriangular square matrix $U$ or $K$ is done by computing the product of inverses of the elementary matrices $E_{ij}(u)$, in the reverse order. Note that the inverse of $E_{ij}(u)$ is just $E_{ij}(-u)$.

# Computing the (left) inverse of a matrix

Having generated an invertible $k \times k$ square matrix $S = UP_1KP_2$, we compute its inverse as $S^{-1} = P_2^{-1}K^{-1}P_1^{-1}U^{-1}$. Computing $P_i^{-1}$ is trivial, and computing the inverse of a unitriangular square matrix $U$ or $K$ is done by computing the product of inverses of the elementary matrices $E_{ij}(u)$, in the reverse order. Note that the inverse of $E_{ij}(u)$ is just $E_{ij}(-u)$.

Now suppose the (left) invertible $k \times l$ matrix $M$ was obtained from the square $k \times k$ matrix $S$ by removing $k - l$ columns $C_{i_1}, \ldots, C_{i_{k-l}}$. Then, to get a left inverse of $M$, we just remove the corresponding rows $R_{i_1}, \ldots, R_{i_{k-l}}$ from $S^{-1}$.

# Suggested parameters

For the hash function $H$, we suggest SHA-512.

For the integer $q$ in $\mathbb{Z}_q$, we suggest $q = 6$.

For the number $n$ of variables, we suggest $n = 64$.

For the dimensions of the matrix $M$, we suggest $k = 10, \; l = 5$.

For the number $t$ of monomials in $t$-sparse polynomials that entries of the unitriangular matrices, we suggest $t = 3$.

# What is the hard problem here?

The (computationally) hard problem that we employ in our construction is finding a left inverse of a given left invertible matrix $M$. That is, solving the matrix equation $XM = I$, where $X$ is the unknown matrix of given dimensions.

# What is the hard problem here?

The (computationally) hard problem that we employ in our construction is finding a left inverse of a given left invertible matrix $M$. That is, solving the matrix equation $XM = I$, where $X$ is the unknown matrix of given dimensions.

If matrices in this equation were considered over a field, then this matrix equation would translate to a system of linear equations (in the entries of the matrix $X$), and therefore would be easily solvable.

# What is the hard problem here?

The (computationally) hard problem that we employ in our construction is finding a left inverse of a given left invertible matrix $M$. That is, solving the matrix equation $XM = I$, where $X$ is the unknown matrix of given dimensions.

If matrices in this equation were considered over a field, then this matrix equation would translate to a system of linear equations (in the entries of the matrix $X$), and therefore would be easily solvable.

In our situation, where matrices are considered over a polynomial algebra, the problem can still be reduced to a system of linear equations, this time in the coefficients of polynomials that are entries of the matrix $X$. If the number $n$ of variables $x_i$ is not too small (we suggest $n = 64$), then the number of different monomials (and therefore the number of unknown coefficients of polynomials) is quite large, so that the relevant system of linear equations becomes intractable.

# Completing a left invertible matrix to an invertible square matrix

Another possible way to find a left inverse of a given left invertible matrix $M$ is to complete $M$ to an invertible square matrix by adding more columns, and then find the inverse of this square matrix (which is relatively easy since one can use determinants).

# Completing a left invertible matrix to an invertible square matrix

Another possible way to find a left inverse of a given left invertible matrix $M$ is to complete $M$ to an invertible square matrix by adding more columns, and then find the inverse of this square matrix (which is relatively easy since one can use determinants).

Perhaps surprisingly, it was a major open problem (Serre's problem) in algebra whether or not any left invertible matrix over a polynomial algebra can be completed to an invertible square matrix by adding more rows (or columns). This problem was answered in the affirmative independently by Quillen and Suslin.

# Completing a left invertible matrix to an invertible square matrix

Another possible way to find a left inverse of a given left invertible matrix $M$ is to complete $M$ to an invertible square matrix by adding more columns, and then find the inverse of this square matrix (which is relatively easy since one can use determinants).

Perhaps surprisingly, it was a major open problem (Serre's problem) in algebra whether or not any left invertible matrix over a polynomial algebra can be completed to an invertible square matrix by adding more rows (or columns). This problem was answered in the affirmative independently by Quillen and Suslin.

The question that matters to us though is that of the computational complexity of completing $M$ to an invertible square matrix. It was shown in [H. Lombardi, I. Yengui, *Suslin's algorithms for reduction of unimodular rows*, J. Symbolic Comput. **39** (2005), 707–717] that there is a relevant algorithm whose complexity is exponential in the square of the number of variables $x_i$. This is yet another reason why the number of variables should not be too small.

# Performance and signature size

For our computer simulations, we used Apple MacBook Pro, M1 CPU (8 Cores), 16 GB RAM computer.

With the suggested parameters, signature verification takes about 0.2 sec on average, which is not bad, but the signature is rather large, about 4,200 bytes on average.

# Performance and signature size

For our computer simulations, we used Apple MacBook Pro, M1 CPU (8 Cores), 16 GB RAM computer.

With the suggested parameters, signature verification takes about 0.2 sec on average, which is not bad, but the signature is rather large, about 4,200 bytes on average.

The size of the private key (the matrix $L$) is about 2,000 bytes, and so is the size of the public key (the matrix $M$).

# Public key encryption

Our digital signature scheme can be easily converted to a public key encryption scheme, as follows.

# Public key encryption

Our digital signature scheme can be easily converted to a public key encryption scheme, as follows.

**Private:** $k \times l$ left invertible matrix $M$, with $k > l$, whose entries are sparse polynomials from the algebra $K$.

**Public:** $l \times k$ right invertible matrix $L$ over $K$, such that $LM$ is the $l \times l$ identity matrix.

# Public key encryption

Our digital signature scheme can be easily converted to a public key encryption scheme, as follows.

**Private:** $k \times l$ left invertible matrix $M$, with $k > l$, whose entries are sparse polynomials from the algebra $K$.

**Public:** $l \times k$ right invertible matrix $L$ over $K$, such that $LM$ is the $l \times l$ identity matrix.

**Encrypting** a message $m$:

1. Convert $m$ to a vector $\mathbf{U} = (P_1, \ldots, P_l)$ of $l$ (sparse) polynomials.
2. Multiply the vector $\mathbf{U}$ by the (public) matrix $L$ on the right to get a vector $\mathbf{V} = (Q_1, \ldots, Q_k)$ of $k$ polynomials from $K$. This vector $\mathbf{V}$ is the encryption of $m$.

# Decryption

Multiply the vector $\mathbf{V} = \mathbf{U}L$ by the private matrix $M$ on the right to get $\mathbf{V}M = \mathbf{U}LM = \mathbf{U}$.

# Thank you