

QRKE: Extensions

G. Brands¹, C.B. Roellgen², K.U. Vogel³

10.26.2015

Abstract

Permutable Chebyshev polynomials (T polynomials) defined over the field of real numbers are suitable for creating a Diffie-Hellman-like key exchange algorithm that is able to withstand attacks using quantum computers. The algorithm takes advantage of the commutative properties of Chebyshev polynomials of the first kind.

We show how T polynomial values can be computed faster and how the underlying principle can further be used to create public key encryption methods, as well as certificate-like authentication-, and signature schemes.

Key words: Chebyshev, polynomial, permutable, real numbers, matrix, multiplication, Diffie-Hellman, RSA, El Gamal, key, exchange, polymorphic, encryption, asymmetric, conference, certificate, authentication, signature, Caley-Hamilton, quantum, computer, decoherence, qubit, Grover, Shor.

1. Fast Computation of T Polynomials of Arbitrary Degree

We add two additional generation algorithms for Chebyshev polynomials to the methods described in our first document [2] which are also described in [3]. The iterative definition of the T polynomials

$$\begin{pmatrix} T_n(x) \\ T_{n+1}(x) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 2 \cdot x \end{pmatrix}^n \cdot \begin{pmatrix} T_0(x) \\ T_1(x) \end{pmatrix}$$

can be used to generate a T polynomial of desired degree, or to compute the values of $T_n(x), T_{n+r}(x)$ by applying x directly into the right side of the equation, and powering the 2*2 matrix with the binary power algorithm with time complexity $O(\log(n))$. The first element of the right vector is directly the desired result.

A generalized C++ code primitive, which is helpful for the understanding of paragraph 11, is

```
template <class T, class S> T power(T b, S e) {
    T res(1); // unity matrix later holding the result
    while (e!=0) {
        if ((e & 1) !=0) res*=b;
        b*=b;
        e>>=1;
    }//endwhile
    return res;
};//end function
```

The template parameters are a primitive 2*2 matrix class with high-precision floating point variables for b , and a large integer for n representing the degree of the T polynomial to compute.

1 Gilbert Brands, D-26736 Krummhörn, e-mail: gilbert(at)gilbertbrands.de

2 Bernd Röllgen, D-35576 Wetzlar, e-mail: roellgen(at)globaliptel.com

3 Kersten Vogel, CH-8055 Zürich, e-mail: kersten.vogel(at)sap.com

The iterative squaring of matrix b may be further improved by using the Caley-Hamilton theorem. The characteristic polynomial of the above generator matrix is

$$\lambda^2 - 2x \cdot \lambda + 1 = 0$$

Assuming $\lambda^r = a_r \cdot \lambda + b_r$ and $\lambda^s = a_s \cdot \lambda + b_s$, we yield

$$\begin{aligned} \lambda^{r+s} &= (a_r \lambda + b_r) \cdot (a_s \lambda + b_s) \\ &= a_s a_r \lambda^2 + (a_r b_s + a_s b_r) \lambda + b_s b_r \\ &= a_s a_r (2x \lambda - 1) + (a_r b_s + a_s b_r) \lambda + b_s b_r = a_{r+s} \lambda + b_{r+s} \end{aligned}$$

So every matrix multiplication in the above algorithm may formally be substituted by a multiplication of two linear polynomials, and substituting λ^2 in the result by the equality derived from the characteristic polynomial resulting again in a linear polynomial representing a higher power of the eigenvalues (formally this is nothing else but $p_1 \cdot p_2 \pmod{\lambda^2 - 2x\lambda + 1}$). Since a matrix satisfies its own characteristic polynomial according to Caley-Hamilton, substituting $\lambda \rightarrow A$ also results in A^n in the power algorithm. When counting the number of calculations, this algorithm is slightly faster than the one above.

The advantage of these algorithms is that the secret index can be chosen arbitrarily, as is the case with the cos/arccos form. The convolution form only allows for secrets decomposing into some small prime factors, which has the impact that the factors get relatively large for a certain security level, and vary in a noticeable range. The accuracy of the floating point variables has to be chosen suitably high, leading to longer calculation times. The matrix powering algorithms allow for smaller secrets with no limits in prime factor decomposition, and therefore allow for shorter calculation times. The lower limit of the size of the secrets should be some decimal powers above the desired security, as we discussed in detail in [2].

We've experimented with the different algorithms and here are a number of results for the complete key negotiation (Alice and Bob).

- a) **Convolution algorithm.** Accuracy 700 digits, 128 bit security, 40 functions * max. 10 iterations. Result: T index interval 580-630, equal digits 90-150, calculation time 90 ms.
- b) **Cos/arccos algorithm.** Accuracy 300/700 digits, 332/930 bit security, T index 200/560. Result: 117/160 equal digits, 11/90 ms calculation time. The second set of values was implemented to get comparable values to the convolution algorithm.
- c) **Matrix algorithm.** The same parameters as in the cos/arccos algorithm were chosen resulting in 13/145 ms calculation time.

All methods can be mixed without any problems. Estimating the calculation times for the above example, the convolution algorithm consumes about 6,000 multiplications, and the same amount of addition operations using the given example. The matrix algorithm consumes about 11,000 multiplications and 5,500 additions for a secret value in the range of the convolution example. The algorithm derived from the Caley-Hamilton theorem ranges in between these values. These estimations coincide with the above measured values. The time complexity of multiplications should be of order $O(1.6)$ when assuming usage of the Karazuba algorithm.

With growing order of T polynomials, the convolution algorithm with time complexity $O(n)$ rapidly becomes inferior when compared with matrix multiplication. When e.g. computing the T polynomial of the order of the 32768th prime number $y = T_{386093}(x)$, $2 \cdot 386093$ multiplication- and 386093 addition operations need to be executed. In contrast to this, matrix multiplication is performed much faster:

700 digits correspond with approx. 2325 bits.

With 8 multiplication- and 4 addition operations per cycle and a 50% chance for two times the number of operations per cycle, 12 multiplication- and 6 addition operations are required on average for each cycle by the matrix multiplication algorithm. In the above example, not more than $2325 \cdot 12 = 27900$ multiplications and $2325 \cdot 6 = 13950$ additions thus need to be executed, which is 27 times less when compared with the convolution algorithm, which executes a for (...) loop over the order of the T polynomial. The table-based version of this algorithm iterates only through half of the order of a T polynomial, but the size of the table that holds coefficients becomes impractical for T polynomial orders exceeding 256.

Due to the fact that matrix multiplication algorithms are neither polymorphic nor purely analytic in contrast to those discussed in our first paper [2], we have to discuss our conjecture about QC safety again. We do this in the last chapter.

2. Agreement Scheme for Conference Keys

Especially in online communications, very often more than two parties participate in a session. Using different keys on two communication legs renders key management more difficult than necessary. Therefore, one key is desirable for all participants. We restrict the following example to three participants only, but the scheme can easily be extended to a higher number of participants.

Alice, Bob, and Chiara agree on a common parameter x and calculate their intermediate values

$$B = T_b(x) , \quad A = T_a(x) , \quad C = T_c(x)$$

using their secrets (a, b, c) . The intermediate values are publicly distributed. The participants are now able to calculate a second set of intermediate values:

$$BA = T_a(B) = T_b(A) , \quad BC = T_b(C) = T_c(B) , \quad CA = T_c(A) = T_a(C)$$

These intermediate values are distributed again. Please observe that it is sufficient that each participant only distributes one value. In practice, a round robin strategy in the indices of the participants may be applied.

Having received the second set, all participants can compute their common secret value

$$ABC = T_{a \cdot b \cdot c} = T_a(BC) = T_b(CA) = T_c(BA)$$

For a practical implementation we have to observe that floating point precision needs to be increased for each additional participant. If N digits of ABC have to be identical for all participants and n is the number of participants and the magnitude of (a, b, c) is of order G , the precision has to be of the order $P = N + n \cdot G$ as a rule of thumb.

Three-party key negotiation is very useful, e.g. for Voice-over-Internet-Protocol applications where a central telephony server and two peers negotiate a shared key.

3. Proof of Knowledge of the Secret Parameter

In a Diffie-Hellman scheme, all parameters are chosen arbitrarily at the time when a communication channel is established. Hence, neither Bob nor Alice can be sure that they communicate directly with each other: Eve can try to mount a man-in-the-middle attack (MITM), which is a very effective method to attack any Diffie-Hellman scheme. Once Eve has mounted the attack, Eve successfully pretends to be Alice to Bob and Eve pretends to be Bob to Alice. Eve simply decrypts all messages exchanged by the two peers and re-encrypts them. As long as Alice and Bob don't check hash values of the exchanged keys, they even do not realize that they've been hacked.

This attack is obsolete once the parameters are bound to a specific person, and use of personalized data is mandatory in the key agreement. Bob may publish the pair $x, y = T_s(x)$ as his public parameters, s being his secret parameter and x being a random high-precision floating point number in the interval $[-1, 1]$.

In order to prove that Bob is in ownership of s , Alice selects a secret value r and sends $T_r(x)$ to Bob. Bob answers by sending $T_s(T_r(x)) = T_{r \cdot s}(x)$ to Alice. Alice computes this value by herself using $T_r(T_s(x)) = T_{r \cdot s}(x)$. Alice can now take advantage of the commutative property of T polynomials and check Bob's authenticity. Only Bob can compute the correct response as only Bob is in possession of the secret parameter s . A man in the middle (Eve) could very well read the plaintext, but the attack would be discovered by Alice and Bob before any critical payload data was exchanged between the two peers.

This method can be used in lieu of a certificate authority of the public parameters of Bob.

In a practical implementation more parameters are needed than in the Diffie-Hellman scheme. The precision of the floating point numbers and the security of the complete scheme can easily be negotiated at key agreement time. If the parameters are fixed, floating point precision may be varied at key agreement time too, but minimum security has to be provided as a parameter in order to assure Alice that the communication takes place at a certain security level. We strongly recommend to fix the floating point precision as well as the number of equal digits of the final secret as an additional parameter as well because Alice can now configure her parameters so that the key agreement will proceed without faults.

4. Interactive Authentication by Publicly Known Parameters

Alice knows the pair $x, y = T_s(x)$ to be Bob's public parameters. Alice and Bob proceed as above using the public parameters of Bob, and individual parameters of Alice, but keep $T_{r,s}(x)$ as their shared secret key. Bob can ensure Alice to have calculated the correct secret by sending a hash value to Alice.

If Bob's parameters are unknown to Alice yet, Alice can fetch a pair x, y from Bob and confirm the values by asking a certificate authority, or by using another secure communication channel (for instance a cell phone, or a physical letter delivered by post). This is just like handling ordinary certificates.

If Bob wants to authenticate Alice by her public values $y' = T_{s'}(x')$ as well, they can follow two strategies:

1. Both can negotiate a second common secret key $T_{s',r'}(x')$ derived with Alice's public parameters, and individual parameters of Bob, and take the product $T_{r,s}(x) \cdot T_{r',s'}(x')$ as their common secret.
2. Bob can take advantage of the El Gamal scheme discussed in the next paragraph to perform a challenge-response negotiation which can only be answered correctly by Alice.

The reader should realize that one secret in each negotiation must be arbitrarily chosen in each agreement in order to prevent replay attacks, or the repeated use of keying material.

5. Public encryption (modified El Gamal scheme)

Bob's public values are again $x, y = T_s(x)$, Alice wants to send a message N to Bob in an encrypted form. The two peers take advantage of an El Gamal scheme. Alice chooses a secret value r , computes $Q = N \cdot T_r(y), R = T_r(x)$ and sends both values to Bob. Bob decrypts the message by computing $N = Q / T_s(R)$. This algorithm is identical to the well-known El Gamal scheme over modular fields, so further explanations are not necessary.

Because this scheme can be applied without any further direct contact of Alice and Bob, Bob has to provide the following working parameters in order to guarantee that the scheme works:

- floating point precision
- number of shared equal digits
- maximum function index of Alice's T polynomial

6. Interactive Signature

Alice wants to prove that Bob is in possession of document N , or that the document N she got from Bob has not been altered. Alice uses Bob's public parameters $(x, y=T_s(x))$ to compute y with the randomly chosen parameter r

$$\begin{aligned} Q &= \text{hash}(N) \\ R &= T_r(x) \end{aligned}$$

and sends Q, R to Bob. The document itself is not sent.

Bob checks whether he is in possession of N by computing $Q = \text{hash}(N)$ for all documents in his database. If he succeeds, he computes

$$C = \text{hash}(T_s(R) \circ N)$$

and sends the value back to Alice. Knowing Bob's public parameter y , Alice can compute C by herself by calculating

$$C' = \text{hash}(T_r(y) \circ N)$$

and checking both values for equality.

In this scheme, only Bob is able to produce a signature because his public values are used. The negotiation cannot be replayed because Alice uses a random r . Bob has to be in possession of the document to generate the signature values. He cannot deliver a signature for a document of which he doesn't have the original version.

The scheme may be extended by encrypting the hash value with an El Gamal scheme.

Since this is an interactive scheme, Bob can deny to deliver a correct signature, either by claiming that he is not in possession of N in the clear, or by delivering a false C . This drawback in comparison with a permanent signature can be compensated by a central registration agency for hash values.

7. Group Secrets

In some applications, for instance in certificate servers, only a group of peers should be able to produce a valid secret value for security reasons. In order to mount such a scheme, a trusted issuer chooses the parameters at random and computes

$$s = \prod_{i=0}^n s_i, \quad y = T_s(x)$$

The s_i are distributed to the participants, (x, y) are public values.

If Alice sends $R = T_r(x)$ to the group, the peers consecutively compute

$$T_{s \cdot r} = T_{s_0}(T_{s_1}(\dots T_{s_n}(R)))$$

The order in which the intermediate results are produced is arbitrary, but they must be produced one after the other. There exists no other way to produce the correct secret.

Due to the multiplicative combination of the secrets, two problems arise:

- The combined secret s may become very large and therefore the required floating point precision as well.
- The more peers cooperate and share their secret, the easier it is to mount an insider attack to unveil the secret s .

8. Partial Group Secrets

To let a subgroup of size $m+1$ of n peers generate a common secret value, a trusted issuer chooses

$$s, p > s, P_m(z)$$

where p is a large prime number and

$$P_m(z) = s + \sum_{k=1}^m a_k x^k \pmod{p}$$

an arbitrary polynomial of degree m . With some arbitrary large integer values the trusted issuer computes

$$z_i, y_i = P_m(z_i) \pmod{p}, 1 < i < n$$

The tuples (z_i, y_i) are distributed among the peers. The calculation uses large integer modulo operation and is, therefore, always precise. $(x, y = T_s(x))$ are again the public values.

To restore the secret, a (second) trusted party (for instance a special hardware device) collects the tuples (z_i, y_i) from at least $m+1$ peers which can be chosen arbitrarily. Having $m+1$ tuples, a polynomial of degree m can unambiguously be reconstructed, for instance the Lagrange's, or Newton's formula. From the construction the first coefficient, or the value $P_m(0)$ is the secret for the T polynomial. The trusted party can now compute the common secret with Alice's intermediate value, and deliver it to the peers.

In order to keep the secret a secret, the trusted party has to be constructed in a way that the tuples are used only once, and then are erased from memory. For the next reconstruction of the secret, the whole process has to be started again.

9. Secret on Behalf of a Group

The intention of a special scheme for a secret on behalf of a group is a possibility to disclose the identity of the acting group member. Alice however does not need to know the identity.

Just like the Partial Group Secret described in the previous paragraph, the trusted issuer sends the three values

$$x, s_i, s'_i = \prod_{\substack{k=1 \\ k \neq i}}^n s_k$$

to each peer. $(x, y = T_{s_i, s'_i}(x))$ are the public values for the group. As can easily be verified, each peer can now compute the common secret with Alice on its own by applying both values s_i, s'_i consecutively:

$$\begin{aligned} y &= T_{s_i}(T_{s'_i}(x)) \\ y_{alice} &= T_{alice}(x) \\ SECRET &= T_{s_i}(T_{s'_i}(y_{alice})) = T_{alice}(y) \end{aligned}$$

The principle of numerics guarantees that the significant digits L are identical, whoever of the peer generates $T_{s_{alice}}(x)$. The region of lost digits by rounding operations however depends on the functions involved, and the consequence is that

$$0 < |T_{s_i, s'_i, alice}(x) - T_{s_k, s'_k, alice}(x)| \leq 10^{-L}$$

The rounding region can therefore be used as a fingerprint of the acting peer. In case of a disclosure, at least $N-1$ group members can verify who the actor was. The differences are however small. Using floating point numbers of 500 digits size, secret indices $r, s \sim 10^{105}$, and 3 parties constituting the peer group, we found using the matrix algorithm:

Number of equal positions between peer group and Alice	~ 300
Region of difference between peers after position	~ 400

A negotiation protocol must guarantee that a sufficiently high number of digits of all parts of the negotiation are preserved to allow for the proof of authorship.

A peer may cheat by multiplying both values delivered to him to get the group secret s as a single value, and thereby circumvent the fingerprint mechanism. In an implementation, further measures have to be taken to force the peers to use the secret values consecutively, for instance by a special hardware device hiding the values from the group members.

10. Limits of the Scheme

As can easily be verified, all proposed protocols are based on the calculation of a common secret parameter using a secret value that was previously unknown to Bob. If the schemes are modified in a way that a party other than Alice is in possession of her second secret value before the communication has started, the secret value may be produced without using Bob's secret. In fact Bob is unable to convince Alice that he has used his secret value.

The reason for this is that

- two secrets cannot be connected by another secret parameter like the factors of the module in RSA encryption.
- combination of secrets is only possible by multiplication (convolution principle). Besides further tricks taking advantage of modulo operations, signatures derived from the El Gamal scheme depend on addition operations besides multiplication operations. Applying only multiplication would unveil Bob's secret to the public.

Therefore a static signature scheme cannot be created using commutative combinations of T polynomials over the field \mathbb{R} ; the signatures have to be generated interactively each time a receiver wants to have a proof of integrity and authorship. The total expense for signatures and authentication functions is therefore higher than for schemes that operate over finite fields because they can be used statically.

As a further drawback, the signer can further decide to deny a signature or authentication he has approved in a former communication, or vice versa. To prevent from situation-dependent conduct, a trusted registration agency needs to be involved resulting in an even higher time complexity.

T polynomials may also be used over finite modular fields, as the composition theorems hold over arbitrary fields. In [3] an RSA scheme is developed using the same construction as in the conventional RSA algorithm by using the secret and public parameters d, e as T polynomial indices. Using pre-published parameters (e, n) , Bob can encrypt a message N by computing

$$X \equiv T_d(N) \pmod{n}$$

and Alice may decrypt it by computing

$$N \equiv T_e(X) \pmod{n}$$

taking X as a signature from Bob. But since (e, n) are publicly known values, the scheme can be attacked using the very same methods as the original RSA scheme, especially by using quantum computers. This scheme is therefore no improvement over the original RSA scheme.

If T polynomials are used over finite modular fields, $p, g, y = T_m(g) \pmod{p}$ are the public parameters as in the El Gamal scheme, m being the secret. The function argument is greater or equal 1 in this case, and the hyperbolic function

$$y \equiv \cosh(m \cdot \cosh^{-1}(g)) \pmod{p}$$

holds in lieu of $y = \cos(r \cdot \cos^{-1}(x))$ over the interval $[-1, 1]$. The resolved equation for m

$$m \equiv \frac{\cosh^{-1}(y)}{\cosh^{-1}(g)} \pmod{p}$$

features no ambiguities due to the hyperbolic function, as was the case in the interval $[-1, 1]$. Using the analytical form of $\cosh^{-1}(x)$, determination of the secret value m can be performed by calculating the discrete logarithm

$$m \equiv \log_{g + \sqrt{g^2 - 1}}(y + \sqrt{y^2 - 1}) \pmod{m}$$

therefore. So again there is no advantage over conventional schemes due to the alleged insecurity of the discrete logarithm.

11. Revision of Quantum Computer Security

In our first article [2] we mentioned Grover's algorithm as a possibility to search for the secret values using a quantum computer. We now show the possible attack path in detail in order to substantiate our conjectures about QC safety of our algorithm. For the details of quantum computing we refer to [1].

We first present a solution for T polynomials over finite modular fields with an n bit integer representation. With recourse to the matrix exponentiation algorithm, the exponent variable m is initialized with an appropriate superposition of all possible states. Since the generation matrix \mathbf{A} is well known (x is a public value), all values of matrix elements in the powers can be hidden as pre-calculated constants in the quantum operations, and no qubit of storage is necessary. The result matrix \mathbf{R} however is initialized with the identity matrix, and is updated in each powering step. For each powering step, a conditional modular multiplication (CMULMOD) representing the if-clause in the algorithm is carried out, the condition given by the exponent q bits.⁴ The intermediate result is mapped on the public value y which can also be done by pure quantum operations because all further values to perform the mapping are constants. After having carried out the complete powering sequence, the complete algorithm is rolled back which means that all steps are inverted until the initial state is reached again. This is necessary principally as quantum operations entangle all variables. The desired information is not concentrated in the variable m alone but in all other variables. Rolling back the whole algorithm sets all other variables but m back to their initial state, and therefore a measurement m becomes meaningful. One mystery of QC is that the control variable itself is part of the entanglement, and not a passive controller of an operation. Rolling back the algorithm also puts back the variable m to its initial state, the superposition of all possible inputs, and a measurement will produce one of these values. The second mystery used in Grover's algorithm is the kind of initialization which gives rise to a phase shift of the exponent value necessary to generate y . So the measurement comprises two signals: a value and its phase. Normally the value is arbitrary and $O(2^n)$ measurements are needed to find the correct value. Using the combination of a value and its phase shift, the complexity to reveal the secret drops to $O(\sqrt{2^n})$.

To store m and the matrix \mathbf{R} , 5 qubit registers with at least n qubits each are required. A 2*2 matrix operation is mathematically performed by computing

$$\mathbf{C} = \mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} \end{pmatrix}$$

To carry out the complete algorithm, the QC programmer can follow two strategies:

1. He can calculate R_{k+1} and maintain R_k until the end of the forward branch of the algorithm and use R_k to erase R_{k+1} only in the roll back operation. All intermediate matrices are kept in qubit registers which makes up $4*n$ values or a memory consumption of $O(n^2)$.

⁴ In a conditional operation, the if-clause of the classical algorithm is not carried out to calculate only one branch of the clause, but both branches are superpositioned with the weight given by the control qubit.

2. He can calculate R_{k+1} , and use this matrix to erase R_k immediately by multiplication of R_{k+1} with A^{-1} , the inverse again hidden in the quantum operations. This would save qubits but double the number of quantum operations.

Qubits are suspected to be the rare good, so strategy 2. will probably be chosen. But as CMUL2 -operations are available at a cost of $O(n^3)$ operations and n matrices need to be calculated, the amount of operations increases to the order of magnitude of $O(n^4)$ (the exact values may be found in [1]).

As can be seen from the formula, multiplication and addition cannot be done “in place”. To carry out an operation is available at cost of two temporary registers of n qubits and the production of the resulting matrix by 8 supplemental registers storing the multiplication results as intermediates. These registers can be reused in the next calculation, and for the estimation of y in each step, but increase the number of registers to implement to $1+4+2+8=15$. To operate on 2,048 bit values, at least 30,720 qubits are needed, and the operations to be carried out sum up to a value that is greater than 2048 cycles * 12 CMUL2 operations per cycle on average * $90*n^3 = 2*10^{16}$. The operation time can be reduced by the calculation of the products in parallel, but this profit must be paid with another 2 register sets per parallel process.

In this study we followed the classical path. In a more sophisticated investigation a couple of improvements may be possible. We refer to [4] on solving the discrete logarithm problem on elliptic curves as an object lesson on how to proceed in general. We don't intensify our studies because encryption over finite fields using T polynomials is only a preliminary step.

In all these values, neither quantum control nor quantum correction is included, which is although broadly assumed to be mandatory in quantum computing and which increases the number of qubits, as well as operations by a factor > 2 . The maximum meaningful number of operations of the most (ideal) stable quantum systems is estimated to be 10^{14} [1]. In [4] it is shown that even Shor's factoring algorithm - as the “leanest” algorithm - comes close to this boundary or exceeds it when factoring modules of today's typical sizes. At a scale like this, the exponential decay of a quantum system has to be considered. If the size of the finite field is increased, the decay becomes the predominant factor in the estimation of the prospects of success, regardless of the complexity of the pure algorithm.

If the second algorithm based on the Caley-Hamilton theorem is considered in the same way, the results may be slightly, but not principally better.

This discussion should only be regarded as a preparation step to investigate an attack on the proposed key exchange using T polynomials over \mathbb{R} . It is not necessary to cope with encryption over finite fields as was mentioned in the previous paragraph because there are some shortcuts to unveil the secret. These do not apply to our encryption model, so we start with the version of Grover's algorithm described in [2].

As we pointed out in [2], the main problem is dealing with floating point numbers on a QC. As a possible workaround based on classical methods, the QC programmer may target fixed point numbers. He knows that the targeted integer m consists at most of n bits and in the same order of magnitude is the size of some of the large values during the calculation. Results may be near 0 and n bits would vanish due to rounding, so a fixed point variable would consist of about n qubits before the decimal point, and $3n$ qubits behind, resulting in a register width of $4n$ qubits. Addition operations don't present problems in principle because they can be inverted, but multiplication operations do. As factors and products each have a $4n$ bit representation, only those combinations of factor digits producing a result in this range can be considered, but not all. The lost digits are the equivalent of the rounding operation in floating point operations. The resulting problem consists in the impossibility of inversion of the multiplication. Having only one factor and the result, the second factor cannot be reconstructed properly (similar problems in general, but not at this scale, appear in the quantum algorithm that solves the discrete logarithm problem on elliptic curves [4]).

The consequence for the QC algorithm: in order to carry out the complete sequence of quantum calculations, all intermediate multiplication results have to be stored. When comparing a QC attack on T polynomials with an attack on 2,048 bit RSA we arrive at the following values:

Attack on algorithm	Number of qubits	Number of operations per try
Shor's algorithm on RSA, $n = 2,048$ bit	$\sim 3n = 6,000$	$\sim 30n^4 = 5.3 \cdot 10^{14}$
Grover's algorithm on T polynomials, $n = 1,024$ Bit (similar runtime as 2,048 bit RSA)	$\sim 1024 \cdot 4 \cdot 12 \cdot 15 \cdot 1024 \cdot 4 = 3.02 \cdot 10^9$	$\sim 1024 \cdot 4 \cdot 12 \cdot 90 \cdot (1024 \cdot 4)^3 = 3.04 \cdot 10^{17}$
Grover's algorithm on T polynomials, $n = 2,048$ Bit	$\sim 2048 \cdot 4 \cdot 12 \cdot 15 \cdot 2048 \cdot 4 = 1.21 \cdot 10^{10}$	$\sim 2048 \cdot 4 \cdot 12 \cdot 90 \cdot (2048 \cdot 4)^3 = 4.86 \cdot 10^{18}$

Table 1: Comparison of QC attacks on RSA and the proposed quantum-resistant public key exchange

Attacking RSA at the cost of EXP_MOD operations is substantially less expensive than attacking 12 times (on average) CMUL2 per cycle with at least $90n^3$ elementary operations with n being 4 times the size due to the necessity to compute fixed point numbers.

Again, in this calculation, quantum error detection and correction are not considered. A more sophisticated review of this algorithm may yield some improvements, but as is shown in [1] and [4], improvements in qubit usage often result in more operations, and vice versa.

The second principle attack algorithm on a quantum computer is Shor's algorithm which allows for the calculation of periods, for instance to determine k in equations like $a + k \cdot b \equiv c \pmod{n}$ which is the crucial point of the insecurity of RSA, DH, and ECC. Using \mathbb{R} as the basic field there are no periods like in modular finite fields. From a theoretical point of view some values for k may be calculated from the cos/arccos form of T polynomials, but this

- 1) is only a purely theoretical consideration with merely no chance of realization, and
- 2) does not break the encryption because a is needed, and the knowledge of k is pointless.

Beyond this, an attack using a modified version of Shor's algorithm has to deal with floating point numbers resulting in similar practical problems as we developed above.

A literature survey shows that no other ideas of principle attack paths using a quantum computer are known today.

Summary

- 1) Shor's algorithm is not suited for an attack on T polynomial encryption algorithm over \mathbb{R} for principle reasons.
- 2) Grover's algorithm is barely suited for an attack, because it only reduces the search space to size \sqrt{n} . Application of this algorithm is easily rendered impractical by using larger n .
- 3) No other principle algorithms that could be suitable to attack the proposed key exchange algorithm are under development today.
- 4) Theoretically an attack algorithm on a quantum computer can be mounted, but the resources being necessary are far beyond today's estimations for the theoretical capabilities of quantum systems. Even if future work might yield more promising attack algorithms, these will most likely not scale on quantum computers.

We therefore renew our conjecture in [2] that the proposed key exchange algorithm is QC safe.

We add some further practical considerations that strongly support this opinion: Shor's attack algorithm on RSA is the most "lean" one. The necessary number of qubits to mount an attack on 2,048 bit RSA including quantum correction is somewhere in the range of 15,000 qubits, of which 4-14 (qubits) are realized [5], depending on the interpretation of the quantum system as a quantum computer device. The state of the art is a factor 1,500 below the minimum requirement to break 2,048 bit RSA. As we have shown, the requirements for an attack on the proposed algorithm is a factor

503,000 above of what is required to break RSA, resulting in a total “distance” of $7.5 \cdot 10^8$ from today's technology.

Even the ideal version of Shor's algorithm including no quantum correction touches the number of operations thought to be possible on ideal quantum systems [1]. Experimental results show strong evidence that the lifetime of quantum systems decreases quickly with a growing numbers of entangled bits [5]. The number of possible operations on a system that is large enough to attack RSA may therefore be a couple of orders of magnitude below the values stated in [1], possibly resulting in the fact that quantum computers cannot even be scaled to the size required to attack RSA. The requirements for an attack on the proposed algorithm are so much far beyond those for RSA, that even ideal QC technology will not scale.

These arguments are in contradiction to discussions in popular science, but having a closer look at these discussions, they restrict themselves on interviews with people claiming that quantum computers may be commercially available tomorrow or the day after tomorrow with public key encryption being history then. A literature survey of reliable scientific publications however reveals no hints where to read about the technological advances backing such claims. The main argument for these opinions in public science discussions seems to be that the NSA and some companies like Google or Lockheed Martin are busy with very classified investigations nobody knows the details of and that this technology may be decades or even centuries ahead of the rest of the scientific world. It is also somewhat strange that quantum cryptography, mathematically proven to be QC safe and being the hype in public science discussions a few years ago, is dead and buried to an extent that it is not even mentioned in today's discussions.

Literature

- [1] G. Brands, Einführung in die Quanteninformatik, 2011, Springer Verlag, ISBN 978-3-642-20646-7

- [2] G. Brands, C.B. Roellgen, K.U. Vogels, QRKE: Quantum-Resistant Public Key Exchange, 2015,
https://www.researchgate.net/publication/282286331_QRKE_Quantum-Resistant_Public_Key_Exchange

- [3] G.J. Fee, M.B. Monagan, Cryptography using Chebyshev polynomials. 2003,
<http://www.cecm.sfu.ca/CAG/papers/Cheb.pdf>

- [4] J. Proos, C. Zalka, Shor's discrete logarithm quantum algorithm for elliptic curves, 2008,
<http://arxiv.org/abs/quant-ph/0301141>

- [5] T. Monz, P. Schindler, J. Barreiro, M. Chwalla, D. Nigg, W. A. Coish, M. Harlander, W. Haensel, M. Hennrich, R. Blatt, 14-Qubit Entanglement: Creation and Coherence, 2011,
<http://mina4-49.mc2.chalmers.se/~gojo71/KvantInfo/LiteratureProjectPapers/14-Qubit%20Entanglement%20Creation%20and%20Coherence.pdf>