Quantum-Resistant Key Encapsulation Mechanism - The RVB algorithm

G. Brands[1], C.B. Roellgen[2]

2017-10-27

Abstract

Classic public-key encryption- and key-establishment algorithms like Diffie-Hellman, RSA algorithms are vulnerable to attacks from large-scale quantum computers, once such machines are reality.

A novel algorithm based on permutable functions and defined over the field of real numbers is proposed. The proposed KEM/key exchange runs fast on universal computers like PCs or smartphones while not allowing for quantum speedup when attempting to break the code.

Key words: Chebyshev, polynomial, real numbers, mantissa, Diffie-Hellman, RSA, key, exchange, encryption, cipher, asymmetric, KEM, symmetric, permutable polynomials, permutable rational functions, Shor's algorithm, Grover's algorithm, quantum, computer, qubit, decoherence, Heisenberg uncertainty, entanglement, post-quantum.

Table of contents

1 The RVB algorithm....2
1.1 Chebyshev Polynomials....2
1.2 Mathematical Properties....3
1.3 T polynomials and Cryptography in Literature....5
1.4 Practical Computation of T polynomial values....7
1.5 Numerics....7
1.6 Representation....8
1.7 Specification of the RVB algorithm....8
1.8 Estimated computational efficiency and memory requirements....12
2 Security....13
2.1 General Remarks....13
2.2 Measurements of Randomness....13
2.3 Brute force sieve using Diophantine Equations....14
2.4 Chosen Plaintext Attacks / Chosen Ciphertext Attacks....15
2.5 Quantum Computer Security....18
3 Further applications of the RVB key exchange....20
3.1 Agreement Scheme for Conference Keys....20
3.2 Certificates and Proof of Knowledge....21
3.3 El Gamal type encryption scheme....21
3.4 Private Authentication and private Signature....21
3.5 Hidden negotiated Signature....22
3.6 Classiscal Static Signature....23
3.7 Group Secrets....24
3.8 Partial Group Secrets....25
3.9 Secret on Behalf of a Group....25
4 Summary....26
4.1 General....26
4.2 Algorithm summary....27
4.3 Security Summary....28

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

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

1 The RVB algorithm

This paper is a continuation of previous publications [16,17,18] started in September 2015.

1.1 Chebyshev Polynomials

The base of the proposed secret shared key negotiation algorithm are the well known Chebyshev polynomials (T polynomials) [2] of the 1st kind which are defined recursively by

$$T_0(x)=1$$
$$T_1(x)=x$$
$$T_n(x)=2\cdot x\cdot T_{n-1}(x)-T_{n-2}(x) \quad \text{for} \quad n\geq 2 \quad \text{with} \quad n\in\mathbb{Z} \;, x\in\mathbb{R}$$

T polynomials are known to feature the semigroup property

$$T_a\left(T_b(x)\right)= T_{a*b}(x)=T_b\left(T_a(x)\right)$$

which enables for use in a Diffie-Hellman-type of algorithm for the public negotiation of a secret key. In short:

- Alice, and Bob agree on a common public value x .
- Alice selects a secret value a at random, sending $T_a(x)$ to Bob over a public channel in the clear.
- Bob selects a secret value b at random, sending $T_b(x)$ to Alice over a public channel in the clear.
- Alice and Bob derive a shared secret value according to the semi group property.

The algorithm can be used over arbitrary fields. Use over modular fields however does not lead to an improvement of security compared with classic Diffie-Hellman or RSA since it results in the same complexity of O(qubits) [11, p.296 ff]. We therefore use it formally over a subset of $\mathbb{R}$ with the mapping

$$T_r:[-1,1] \rightarrow [-1,1]$$

Within this restriction the alternative definition

$$T(x)=\cos(a*acos(x))$$

also holds, which has to be considered in the discussion of QC security. T polynomials are rapidly oscillating functions in the interval $[-1,1]$, $T_a(x)$ having a simple roots at

$$x_{k,a(0)}=\cos\left(\frac{2k-1}{2a}*\pi\right)$$

and extrema at

$$x_{k,a(e)}=\cos\left(\frac{k}{a}*\pi\right)$$

When selecting a in a magnitude that is common for cryptographic applications, i.e. $2^{300}\leq a$, $T_a(x)$ yields a strong pseudorandom number. Given $x, y=T_a(x)$, there is no efficiant way to compute the secret parameter a from $T_a(x)$, even if a quantum computer operating with the number of qubits that are sufficient to easily break classic Diffie-Hellman or RSA is used for the attack, as will be shown.

Due to the fact that only numbers that are part of the subset $\mathbb{Q}$ can be represented on a classical computer, we have the further restriction that only rational numbers from the interval $x\in\mathbb{Q}: -1<x<1 \;\wedge\; x\notin\{-1/2,0,1/2\}$ are used. This is another important fact that affects classic- and quantum computer security as can be concluded from the mathematical properties.

1.2 Mathematical Properties

From the above properties the following theorems can be derived:

Theorem 1. If $x \in \mathbb{Q} : -1 < x < 1 \ \wedge \ x \notin \{-1/2, 0, 1/2\}$ there exist no weak arguments.

Proof. Except for the excluded values, T polymials only have roots and extrema in $\mathbb{R} \setminus \mathbb{Q}$. Roots and extrema on the other hand are fixpoints in calculations, and therefore weak input values. Taking x from the defined interval, no fixpoint can be reached during the generation of a T polynonial value.

Theorem 2. The only way to calculate the shared secret of Alice and Bob within the boundary condition $x \in \mathbb{Q}$ is to use the secret values of both.

Proof. Assume there exists an index $r \neq a$ fulfilling $T_r(x) = T_a(x)$. Thus $P(x) = T_r(x) - T_a(x)$ is a polynomial of degree $max(r, a)$ having at most $max(r, a)$ roots. On the other hand $y = T_b(x)$ is a strong pseudo random number, b being as well an arbitrary random number. Thus fulfilling also $T_r(y) = T_a(y)$ is possible if and only if y is a root of $P(x)$ as well, which is (nearly) impossible for a random argument. It can therefore be concluded that

$$\left(T_r(y) = T_a(y) \Leftrightarrow P(y) = 0 \right) \Rightarrow r = a .$$

This conclusion might appear to be weak in the strong mathematical sense up to this progression of the proof. This gap will be closed in theorem 4.

Theorem 3. There exist no side channel attacks using alternative forms of T polynomials within the boundary condition $x \in \mathbb{Q}$.

Proof. From $y = \cos(a * acos(x))$ with known values x, y, Eve may calculate $a = \frac{acos(y)}{acos(x)}$ to derive a. Taking the ambiguity of the trigonometric function into account, Eve‘s solution is however ambigeous to

$$a_k(x) = \frac{\pm \arccos(T_a) + 2 \cdot \pi \cdot k}{\arccos(x)} = \pm d + e \cdot k \quad , \quad k \in \mathbb{Z} \ , \ d, e \in \mathbb{R}$$

Eve now has to look for solutions k satisfying $a(k) \in \mathbb{Z}$ of this equation which exist per precondition.

Case 1: let $d, e \notin \mathbb{Q}$ and $\pm d + e * k \in \mathbb{Q} \ \wedge \ \pm d + e * l \in \mathbb{Q}$ with $|k - l| > 0$ being minimal. Then

$$e * (k - l) \in \mathbb{Q} \ \vee \ \pm d + e * \left(\frac{k - l}{2} \right) \in \mathbb{Q} \quad \Rightarrow \quad k = l$$

So there exist one, and only one solution in this case. There is no known analytical solution for equations of this type.

Case 2; let $d, e \in \mathbb{Q}$. From $acos(y) = \pi \cdot y_z / y_n$, $acos(x) = \pi \cdot x_z / x_n$ the diophantine equation

$$k \cdot 2 \cdot y_n \cdot x_n - n \cdot y_n \cdot x_z = - y_z \cdot x_n$$

can be derived which has solutions if and only if

$$y_n \leq gcd(2 \cdot y_n \cdot x_n, y_n \cdot x_z) \mid y_z \cdot x_n$$

or, since all fractions are truncated to the smallest value per default, if $x_n = l * y_n$. Trigonometric functions definitely don‘t allow for obtaining this feature. Therefore there exists no algebraic solution for this problem.

From both paths we obtain the assertion that there is one and only one solution to the problem with neither analytic nor algebraic possibilities to identify the solution.

Theorem 4. If $T_a(x)=T_r(x)$, $a\neq r$ then $x\notin\mathbb{Q}$.

Proof. This is one of the rare cases in mathematical deduction where a proof can be based on a few exemplary calculations. T polynomials intersect in the interval $[-1,1]$, and as can easily be verified experimentally, the intersections of T polynomials show periodic character (Fig. 1).

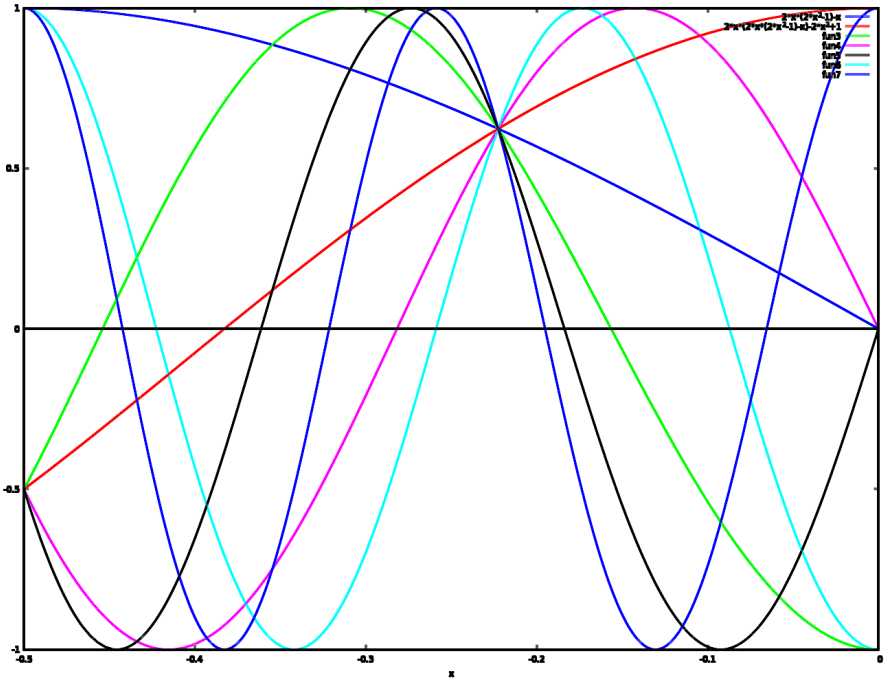

Fig. 1: T polynomials sharing an intersection

a) The image shows an intersection of $T_3,T_4,T_{10},T_{11},T_{17},T_{18,}T_{24},\ldots$ clearly indicating the periodic character in the degree (index). From theorem 3 we already know that there are no periods if $x\in\mathbb{Q}$, hence intersections are not rational which closes the proof of theorem 2.

b) The points of intersection are defined by

$$\cos(a*acos(x))=\cos((a+i*k)*acos(x)) \quad , \quad i\in\mathbb{N}$$

or

$$\cos(a*acos(x))=\cos((a+t+i*k)*acos(x)) \quad , \quad i\in\mathbb{N}$$

with k being the period (in our example calculation $k=7$), and t a non-periodic distance (in our example $t=1$). Comparing the arguments of the cosine function in these expressions with the arguments in the periodic equality $\cos(z)=\cos(\pm z+j*2*\pi)$, $k\in\mathbb{Z}$ just leaves for $acos(x)$ the possible values

$$acos(x)=\frac{l}{m}*\pi$$

with l,m being some integer numbers that can be calculated from the values selected for t,i,k . Hence the intersections coincide with the extrema, or roots of the T polynomials (Fig. 2 as an example showing the intersection of T_3,T_4 , and the maxima of $T_{7,}T_{14}$).

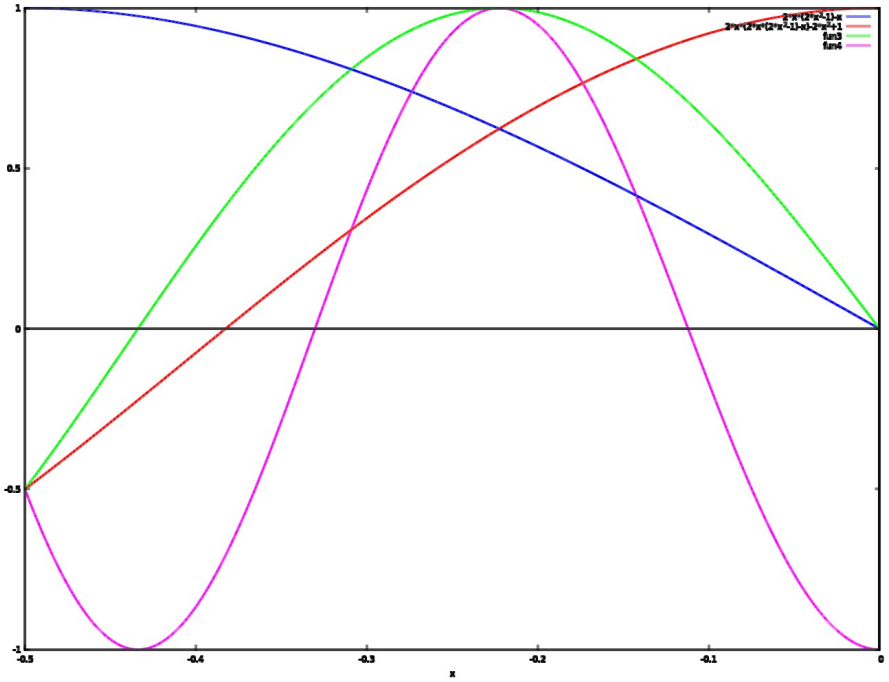

Fig. 2: Relation of intersections and extrema

1.3 T polynomials and Cryptography in Literature

In a paper of Bergamo et al [5] the authors claim that Chebyshev polynomials can easily be attacked. They suggest that an integer number $r'=r$ can be found with the property $T_r(x)=T_{r'}(x)$, which breaks the encryption, and that the inverse of the cosine function can be used to discover the secret r as well. A numerical example serves in the paper [5] as „proof“. Although being able to completely refute these assumptions in chapter 2.3 „Brute force sieve using Diophantine Equations“, Bergamo et al [5] may be misused by critics as a simple way to cast doubt on the security of using Chebyshev polynomials of the 1st kind in cryptography.

Please note: the following statements are not our opinion; they are strictly mathematical facts. In our mathematical section (chapter 2) we present proofs for them. Mathematics has nothing to do with democracy. There is no room to „vote“ for other possibilities if a theorem is proven.

Within the working range of the RVB algorithm we have to state:

- The conclusions of Bergamo et al. are mathematically wrong. An alternate value for the secret resulting in the same outcome of the T polynomial does not exist.
- The methods to uncover the secret do definitely not work. A solvable diophantine equation derived from the inverse cosine function does not exist.
- The numerical example presented as a „proof“ is definitely false.

The main reason for the false conclusions is the assumption that the properties of T polynomials are the same over $\mathbb{Q}$ and $\mathbb{R}\setminus\mathbb{Q}$. In fact, this is definitely not the case. There exist very essential differences, as we have proved in chapter 1.2 (theorem 3). In practice (on a computer), one can only work with a subset of $\mathbb{Q}$ because there exist no complete representations of elements $\in\mathbb{R}\setminus\mathbb{Q}$. If this is not considered, fatal conclusions like those of Bergamo et al [5] are the consequence.

The analysis of the numerical example in [5] serving as proof, is really embarassing. Critics tend to believe the figures stated in the paper. It is consequently necessary to present the complete calculation. In [5] on page 9 the authors compute for x=0.64278761, s=106000:

$T_s(x)=0.173648178$, which is only correct up to the 4th decimal place. The correct value is:

$T_s(x)=0.173690893088351999313561585137392592402118252992497530447245608667144329398558104201 8$

For r=81500, the authors compute.

$T_r(x)=-0.939692621$. The correct value is

$T_r(x)=-0.939681214133139986952561667773533078920555118423372130350500942541644313557458735 70$

The value stated by the authors is again only correct up to the 4th decimal place.

According to the authors, $T_r(T_s(x))=0.766044443$. In reality, **and by using hundreds of decimal places**,

$T_r(T_s(x))=-0.953937231681424364181039608577097826555875775999817165094836639458743954051558 9$

Crosscheck:

$T_s(T_r(x))=-0.953937231681424364181039608577097826555875775999817165094836639458743954051558 9$

The authors thus have not matched a single digit of the true result.

The authors further claim that $T_4(T_s(x))=T_{32}(T_s(x))=0.766044443$. That's a total miss as

$T_{32}(T_s(x))=0.174699043162826814881760162282552917334978619923115135525972345556681768508068$

and

$T_4(T_s(x))=0.173516805550237908887906125847960680920839633665698416028548148442461747961120 3$

share only two decimal places.

The following C++ source code snippet was used to compute the exact values:

```
mpreal x,Tr,Ts,Trs,Tsr;
x=0.64278761;
mpz_class r=106000;
mpz_class s=81500;
Tr = calculate(x, r);
Ts = calculate(x, s);
cout << "Tr(x)=" << endl << Tr.toString() << endl;
cout << "Ts(x)=" << endl << Ts.toString() << endl;
Trs = calculate(Ts, r);
Tsr = calculate(Tr, s);
cout << "Trs(x)=" << endl << Trs.toString() << endl;
cout << "Tsr(x)=" << endl << Tsr.toString() << endl;

mpreal T32s,T4s;
r=32;
T32s = calculate(Ts, r);
r=4;
T4s = calculate(Ts, 4);
cout << "T32s(x)=" << endl << T32s.toString() << endl;
cout << "T4s(x)=" << endl << T4s.toString() << endl;
```

The authors got caught in multiple traps, notably in running out of good decimal digits and by oversimplifying the use of diophantine equations, which are further discussed in chapter 2.3. [5] is probably the reason why many people strongly believe that T polynomials of the first kind can easily be attacked, and retain this believe without noticing that

overwhelming facts prove the opposite. This is not unprecedented in history: Several hundred years ago everybody strongly believed that the sun rotates around the earth.

1.4 Practical Computation of T polynomial values

To come to a useful formula for the fast evaluation of arbitrary T polynomials with large degrees, the recursive definition can be expressed in matrix notation:

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

Since $T_0(x)=1, T_1(x)=x$, this formula allows to compute the functional value of the T polynomial with degree n by a simple matrix exponentiation. The exponentiation itself can be executed by the classical binary exponentiation algorithm with complexity $\log(n)$.

The computation may further be improved by using the Caley-Hamilton theorem. Thecharacteristic 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 $p_1\cdot p_2 (mod\, \lambda^2 - 2x\lambda + 1)$). 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 may be slightly faster than the one above depending on the implementation.

1.5 Numerics

Using large indices results in bulky rationals which cannot be handled. Therefore floating point numbers with fixed length have to be used which brings in another handicap that is particularly important for quantum computer calculations.

From the definition formula of T polynomials we know that the integer coefficients a_k of $T_r(x) = \sum_{k=0}^{r} a_k * x^k$ span an interval $1 \le a_k < 2^k$ in magnitude, yielding up to a real number $-1 < y < 1$ in the evaluation of the argument. Therefore we have to cope with a massive loss of digits during the calculation.

Floating point numbers have a fixed length and are always normalized to the most significant bit. If two similar numbers are subtracted from each other, the most significant bits of the result are zero, but the remainder is shifted left until the highest bit is (formally) set:

$$1.1234567\,xxxxxxx - 1.1234567\,yyyyyyy = 0.000000\,zzzzzzz \rightarrow z.zzzzzz\,00000000$$

In this simplified example, two numbers of 15 digits in length each result in a new number with formally 15 digits, but only 7 of them having physical significance, the rest being added during the nomalization operation.

During the key negotiation process between Alice and Bob each using secret numbers with n decimal digits, it must be expected in practice due to these effects, that the least significant $2*n$ digits in the final key are not identical on both sides. Therefore the length of the floating point numbers for a calculation with secrets of size n decimal digits

and a shared key of at least n decimal digits expected to be equal in the results on Alice's side as well as on Bob's, the register size must be set to $3*n$ decimal digits at minimum.

Remembering our remark on theorem 3, these effects were obviously not accounted for in [5]. The authors mutually used mid range secret values and standard IEEE 754 double floating point variables. Only approximately two matching decimal digits in the shared secret of Alice and Bob from a total of 16 digits could be observed.
The number of actually matching decimal digits on both sides differs between key exchanges and this effect can be exploited to yield longer shared keys. Instead of selecting a fixed length, both peers can exchange hash values of the shared key with e.g. $n+5$, $n+10$, .. decimal digits and mutually agree on the longest key with matching hash values. The resulting key exchange is much more complicated to implement and it is generally better to select a longer register size if longer keys are required in an application.

The shared public parameter x is potentially susceptible to producing secret values -1, -0.5, 0, 0.5 and 1.0. A very elegant way to harden the proposed key exchange algorithm against this is to use numerics: If one of the participiciants or a third party is able to provide an x-value of a root or an extrema with full precision of the floating point representation, this may be used in some malicious way (although we arrived at no idea how to realize a fraud). An intermediate or shared key value may consist of a large number of zeros or nines in the mantissa leading to a widely known result or preventing to arrive at a shared secret value at all.

It is sufficient to cut the x-value to the targeted precision of the shared key and to fill the remaining digits with zero. Both secret values of the two peers will be far away from any root or extremum and the desired secret shared secret will appear as the result without significant loss of accuracy.

Roots and extrema have been explicitly excluded in chapter 1.1. Another possibility to identify shared public parameters x that result in roots or extrema is to explicitly check the computed secret values. Fraudulent action can although only be prevented by providing a truncated shared public x .

1.6 Representation

In order to exchange values between participiciants using software from arbitrary manufacturers, the floating point values have to be represented in a universal manner. The ASCII representation as a human-readable number string with base 10 is part of every numeric library, and is also a valid representation in ASN.1 notation, which is widely used in protocol definitions. It thus appears to be advantageous to use this human-readable notation.

Exchanged values may become very small. Different libraries may represent small values in different string forms, for example

$$0.00056.. \equiv 5.6...e-004$$

To guarantee compatibility of different implementations to some extent, it is recommended that values are represented as stings starting with $\pm 0.xxxx$ up to a certain number of zero characters after the dot before switching to exponential representation. Most libraries support this for up to 16 digits (double precision).

In order to derive a shared binary string, a cryptological secure hash function, for instance SHA-3, should be used.

1.7 Specification of the RVB algorithm

The proposed key agreement algorithm implemented as Key Encapsulation Mechanism for the participation in the NIST PQC project works in analogy with the DH/El Gamal algorithm and is specified as follows:

Alice and Bob agree on the targeted length of the shared key (e.g. 256 / 384 / 512 bit = 32 / 48 / 64 byte). The system calculates the secret size *n* (100 / 150 / 200 decimal digits) and the length of the register size that holds the mantissa. This register size must be set to $3*n$ decimal digits at minimum due to the loss of digits by rounding operations.

Alice and Bob subsequently agree on a publicly known real number x with a total number of decimal digits beetwen *n* and 2**n*, and choose some large secret integer values a and b with *n* decimal digits at random and compute:

$$T_a(x) \ , \ T_b(x)$$

Alice and Bob then exchange these values via an insecure channel.

The mutual secret key is computed by combining the two functions

$$T_a(T_b(x))=T_b(T_a(x))=T_{a*b}(x) \ .$$

Alice's result $T_a(T_b(x))$ and Bob's result $T_b(T_a(x))$ should theoretically be identical. Due to inevitable rounding errors, typically the first third of the bits of the mantissa equal on both sides. The remaining two third of the bits need to be cut away. Only the most significant decimal digits of the targeted length of the shared key (e.g. *n*=100 decimal digits + a certain additional amount to account for bias of the first decimal digits) of the mantissa thus remain and constitute the raw shared key.

The raw shared key is compressed by a hash function like SHA-3 and thus results in the binary shared key of desried length. In contrast to DH or RSA, which both yield 100% identical results on both sides, only the first third of the bits of the mantissa can be used for this operation.

The iterative generator is used to directly compute the function value of T polynomials with index n featuring a similar number of decimal digits:

$$\begin{pmatrix} y_n \\ y_{n+1} \end{pmatrix}=\begin{pmatrix} 0 & 1 \\ -1 & 2x \end{pmatrix}\begin{pmatrix} y_{n-1} \\ y_n \end{pmatrix}=\begin{pmatrix} 0 & 1 \\ -1 & 2x \end{pmatrix}^n\begin{pmatrix} 1 \\ x \end{pmatrix}$$

The powering of the 2*2 matrix can be executed with the binary power algorithm with time complexity $O(\log(n))$. The first element of the right vector is directly the desired result after multiplying the result matrix with the vector $\begin{pmatrix} 1 \\ x \end{pmatrix}$.

This is a generalized C++ code primitive for fast exponentiation with the variable b being an object of a matrix class and e being the exponent:

```
template <class T, class S> T power(T b, S e){
    T res(1); // the unity matrix later holds the result
    while (e!=0) {
        if ((e & 1) !=0) res*=b;
        b*=b;
        e>>=1;
    }//end while
    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 (index) n of the T polynomial to compute. The matrix and vector class is defined by

```
template <class T> struct v2 {
    T a1,a2;
};

template <class T> struct m22 {
    m22()
    {
        a11=a12=a21=a22=0;
    }
    m22(size_t n)
```

```
        {
            a11=a22=n;
            a12=a21=0;
        }
        m22& operator*=(m22 const& m)
        {
            m22 c;
            c.a11 = a11*m.a11 + a12*m.a21;
            c.a12 = a11*m.a12 + a12*m.a22;
            c.a21 = a21*m.a11 + a22*m.a21;
            c.a22 = a21*m.a12 + a22*m.a22;
            *this=c;
            return *this;
        }

        v2<T> operator*(v2<T> const& v) const
        {
            v2<T> res;
            res.a1=a11*v.a1+a12*v.a2;
            res.a2=a21*v.a1+a22*v.a2;
            return res;
        }

        m22& operator=(size_t i)
        {
            a11=a22=i;
            a12=a21=0;
            return *this;
        }
        T a11,a12,a21,a22;
    };
```

We favourite C++ as programming language because it is available on almost all platforms and it is known to produce fast executable code from best readable source code. The reference implementation in ANSI C provided for the NIST PQC project follows the exact same modus operandi.

The template parameters in the preceeding code are long float, and long integer variables with the length adjusted to the desired security level. Instead of reinventing the wheel once again, we decided to use standard multi-precision libraries for our implementation. The GMP and MPFR libraries used for the NIST reference code can hardly be included as source code in a project, but must be used as static or dynamically linked libraries, which might be a disadvantage in some development projects. We therefore make a supplemental C++ implementation available on request, which uses a compact, but less efficient library with the source completely included in the project.

A separate implementation of the Caley Hamilton algorithm as a substitute for binary exponentiation is omitted because it would only provide a minor advantage in execution time.

For the actual form of the algorithm, KEM has been selected (cf. footnote on page 23).

The function **int crypto_kem_keypair(unsigned char *pk, unsigned char *sk)**

selects a long integer number at random named “common“ as well as a long integer random number for the secret key sk and calculates the public key pk by computing

$$pk = T_{sk}(common) \quad .$$

The common value is selected at random with exclusion of values starting with 0.00, 0.50, and 0.99. As many as 2*CRYPTO_SECRETBYTES decimal digits are generated. Random numbers are thus automatically selected to be far away from critical values described in section 2.4. The limitation of the width guarantees that the value is not near any root or extremum. The sign is omitted for the sake of simplicity because the function graphs are either point symmtric or axis symmetric.

The variable sk contains the secret key as a decimal ASCII string.

The variable pk contains the public key as a decimal ASCII string, followed by the common value as e second decimal ASCII string starting directly after the closing 0 character of the first string.

Bob creates the key pair by calling the above function and sends the public key to Alice. Alice sets the shared secret ss, an array comprising 32, 48 or 64 bytes, to the desired value. *ss* will later contain the secret shared by sender Alice and receiver Bob.

Alice subsequently calls **int crypto_kem_enc(unsigned char *ct,unsigned char *ss,const unsigned char *pk).**

The function computes

$mask = T_{tsk}(common)$ with the large integer number *tsk* being selected at random by Alice.

The function subsequently computes $y = T_{tsk}(pk)$, truncates the decimal string representing *y* to approx. a third of its total length and passes the resulting string $y_{truncated}$ to a hash function:

$shared = SHA3(y_{truncated})$.

Finally the shared secret *ss* is XORed with *shared*

$msg = shared\ xor\ ss$

and the resulting value *msg* is sent together with *mask* to the receiver Bob (both values are concatenated and form the parameter *ct*).

The variable *ss* is assumend to contain the secret in binary form, the variable *pk* is Bob‘s public key explained above.

The variable *ct* contains the encrypted secret in binary form, followed byAlice‘s public key for decryption as decimal ASCII string, which directly starts after the secret bytes, and is followed by the common value (same as Bob‘s) as decimal ASCII string.

Bob calls **int crypto_kem_dec(unsigned char *ss,const unsigned char *ct,const unsigned char *sk)**

in order to decrypt *msg*, the function separates the values *msg* and *mask* in *ct* and computes

$y_b = T_{sk}(mask)$, truncates the decimal string representing y_b to approx. a third of its total length and passes the resulting string $y_{btruncated}$ to a hash function:

$shared_b = SHA3(y_{btruncated})$.

Finally the value *msg* is XORed with $shared_b$

$ss = shared_b\ xor\ msg$

and the value that sender Alice has selected for *ss* is decrypted.

The reference code is configured to handle 32 byte secrets. This can however be scaled by selecting other secret sizes in api.h before compiling:

```
#define CRYPTO_BYTES 32
//#define CRYPTO_BYTES 48
//#define CRYPTO_BYTES 64
```

In order to encrypt 384 bit, or 512 bit secrets instead of 256 bit, simply uncomment the respective #define statements.

1.8 Estimated computational efficiency and memory requirements

The proposed key agreement algorithm executes as 64 bit machine code on a single processor core of an Intel Core i7-6700K running at exactly 4.00GHz at the following speeds:

Targeted accuracy [decimal digits]	Floating point precision [decimal digits]	Execution time of ANSI C implementation [ms] using GMP and MPFR numeric library (Compiler: GCC 4.9)
100	366	6
200	655	22
300	963	59

Table 1: Execution time of an Intel Core i7-6700K running at 4.00GHz of both halves of an RVB key exchange in a single thread using different numeric libraries

Execution time comprises the complete key exchange for both participants, namely the generation of the two public keys $y_r = T_r(x)$ and $y_s = T_s(x)$, as well as the generation of both secret keys $z = T_r(y_s)$ and $z = T_s(y_r)$. In any real-world application, only one public and one secret key is generated by an endpoint. As a matter of consequence, the figures that are stated for execution times in all the tables is this paragraph need to be divided in half in order to yield the execution time for a single endpoint.

Optimizations are nearly impossible, as long as linkable standard libraries are used because these libraries themselves use system dependent optimization strategies which cannot be controlled by the application developer. Our optimized implementation therefore is identical to the reference implementation. Using small and directly included libraries in source code as for the C++ implementation mentioned above, optimization strategies using for instance OpenMP can be applied successfully.

Execution time is especially critical in server applications. Servers are in most cases likely to emit a certificate, so that only one operation, namely calculating the shared secret, will be necessary. The GMP/MPFR library is rather complex, large and highly system dependent, but should be available on most server hardware, and by using this library, the central part of the key exchange calculation yielding a key with at least an equivalent strength of a 256 bit symmetric cipher is executed in only 3 .. 4 ms on a single CPU core. In client applications, however, time will not be critical, so that other libraries can be favored as their source code can be directly compiled together with the source code of the application program, which eases platform independent programming.

Memory consumption has it's peak in the calculate() function. On each side of the key exchange (Alice/Bob), the following RVB parameters must be held in Random Access Memory:
mpz_t: the secret value and the exponent in the exponentiation algorithm two times as an intermediate =3 variables
mpfr_t: the common x , the keys y_a and y_b, the result res, the m22 matrix m containing 4 mpfr_t, the vector v containing 2 mpfr_t, 8 mpfr_t in the power() function and up to 16 mpfr_t in mult_m22_m22() depending on the level of optimization, resulting in 34 mpfr_t variables.

The following table depicts the minimum memory requirements for the three different target accuracies in bit:

Variable	Targeted accuracy: 100 decimal digits	Targeted accuracy: 200 decimal digits	Targeted accuracy: 300 decimal digits
mpfr_t	34 * 1,472	34 * 2,432	34 * 3,456
mpz_t	3 * 1,344	3 * 2,304	3 * 3,328
Sum [bytes]	54,080 bit = 7 k	89,600 bit =11 k	127,488 bit = 16 k

Table 2: Memory requirements of the proposed key agreement algorithm on the NIST PQC reference platform

2 Security

2.1 General Remarks

The proposed algorithm is comparably simple, which is helpful for security assessment. Security estimations are based purely on the mathematical properties of the algorithm, which also holds for quantum computer security. The underlying mathematics leave no room for any attacks that target design flaws. Simplicity might be one of the key advantages of the algorithm. We show that attacks proposed in earlier literature do not work.

2.2 Measurements of Randomness

As proven in the previous mathematical section, there exists (today) no efficient way to identify the secret index from the publicly known values $x, T_a(x), T_b(x)$. Taking into account that someone might conceive an algorithm that performs better than brute force, we assume that in order to assure 256 bit security, 100 decimal digits (about 330 bit) should be sufficient, resulting in floating point operations with register sizes of approximately 1,200 bit.

Classic RSA and DSA schemes require key sizes of at least 2000 bits [15, Table 1.2] in order to feature a security level of 128 bit for symmetric ciphers. The proposed algorithm is much faster than DH or RSA.

Alice and Bob's shared key $T_{ab}(x)$ has been tested experimentally for randomness with arbitrary values for the parameters *a*, *b* and x as well as with chosen values with small known differences between the values using the Diehard battery of randomness tests [13].

The shared numeric secret string deviates from randomness in the first 3-4 binary digits because the gradient of the trigonometric function varies strongly between root and maximum. Our tests show that the remaing binary digits of the mantissa exhibit proper pseudo random behaviour with no bias.

Uneven distribution due to the 3 leading bits can be observed in the following tests:

- Overlapping-Pairs-Sparse-Occupancy: digits 1 .. 10 and 23 .. 32
- COUNT-THE-1's TEST for specific bytes: digits 1 .. 8, 2.. 9, 3 .. 10
- SQEEZE test: p-value=1.000000
- CRAPS TEST: p-value for no. of wins: .002047, p-value for throws/game: .992118

When removing the 3 leading decimal digits, all tests were passed without the slightest deviation from perfect randomness. The positive test result leads to the conclusion that the quality of Alice's selection of a random x can feature imperfect randomness without affecting security of the key exchange.
As is good practice for DH orRSA implementations, the mutual secret value is to be transformed to a secret encryption key by hashing an appropriate number of decimal digits. In the reference implementation the entire string containing the sign, comma and the mantissa is supposed to be fed into the compression function for the sake of simplicity and because there are no negative side effects of doing so.

2.3 Brute force sieve using Diophantine Equations

Although we have proven in theorem 3 that a direct solution of the inverse of the cosine representation of Chebyshev polynomials

$$r = \pm\frac{\arccos(y)}{\arccos(x)} + \frac{2\cdot\pi}{\arccos(x)}\cdot k = \pm d + e\cdot k \quad \text{with} \quad r, k \in \mathbb{Z} \;,\; d, e \in \mathbb{R}$$

cannot be used to yield an equation that allows to evaluate the parameter r, we follow the route given in [5] to investigate whether an approximation of r can be guessed with some degree of fuzziness. This could open a significantly rapid way to determine the correct value – much faster than pure blind brute force.

The goal is to find an integer number k so that all digits of the fractional part of both real numbers cancel each other out. Abstractly worded, the sieve is on for a solution which satisfies

$$\pm d \equiv k\cdot e \pmod{\mathbb{Z}}$$

whereat $a \equiv b \pmod{\mathbb{Z}} \;\Leftrightarrow\; a - b \in \mathbb{Z}$ means that the difference of two real numbers a, b must be in the subring of integer numbers. This appears to have some similarities with modular arithmetic, which is frequently used in cryptography and which requires ideals over a ring. Integer numbers are although everything but an ideal of real numbers. They are only a subring of real numbers, and therefore modular arithmetics cannot be applied directly.

The method of resolution in [5] consists of the multiplication of a real number by a sufficiently large integer number $M = 10^m$ (the representation as decimal power is not mandatory but eases analysis so that it is possible to stick to the assumption without loss of generality), taking the integral part for the further calculations. By doing so, the equation is transformed into a diophantine equation:

$$\pm[d\cdot M] \equiv k\cdot[e\cdot M] \pmod{M}$$

The brackets [..] represent the floor function (mapping of a real number to the smallest adjacent integer), or in an equivalent form not using a modulus

$$k\cdot[e\cdot M] + n\cdot M = \pm[d\cdot M]$$

In order to be solvable,

$$gcd([e\cdot M], M) \mid [d\cdot M]$$

must be fulfilled. In order to arrive here, a multitude of proven algorithms exist since the ancient world: the (extended) Euclidean algorithm, the continued fraction method, the Euler method and Euler's theorem with the aid of the Chinese Remainder Theorem. Having the special solutions (k', n') depending on the solution method used, all other solutions are of the form

$$k(z) = k' + M\cdot z$$
$$n(z) = n' + [e\cdot M]\cdot z$$

with arbitrary $z \in \mathbb{Z}$, and the expectation is to find the exact secret r among the $n(z)$ series.

Whereas in [5] the proposed method was used to detect the secret directly (we already know that this is a fake), the idea was to use the values as the centers of a linear grid, and search for the correct value starting from these values by gradually increasing the radius of the search interval. As for brute force, such a sieve must hit the value somehow.

To sum up the results:

- Very small values for M produce a fine grid that is not useful for a sieve.
- Very big values for M produce grid points not even being of the same magnitude as the secret value.
- A small interval of M values produces a manageable number of grid points in the appropriate interval but these values are randomly distributed, meaning the values of M have nothing in common with values of M'.
- Empirical evaluations show that the secret value is randomly distributed between the grid points.

This attack consequently performs not better than a pure brute force evaluation of possible values for r.

2.4 Chosen Plaintext Attacks / Chosen Ciphertext Attacks

In this chapter three different attacks are described. The first two (IND-CPA and IND-CCA2) depend on a public encryption scheme like El Gamal. The El Gamal scheme for the proposed key exchange algorithm is defined later in the paper (see section 3.4). The third attack applies to the basic algorithm.

The IND-CPA Scheme

The key exchange proposed in this paper is primarily intended for ephemeral-only key establishment. The proposed key exchange must consequently provide semantic security with respect to chosen plaintext attack (IND-CPA).

This is how the game between an adversary and a challenger is defined:

The adversary shall have unlimited access to a probabilistic polynomial time Turing Machine – the „oracle“.

Challenger Alice selects the secret key a and the public parameter x at random and computes the public key $T_a(x)$. Alice retains the secret key a and sends x and $T_a(x)$ to Bob.

The adversary (Bob) now is free to perform any number of operations in polynomial time to generate two different messages $b_{1,}b_2$, and to send them to the challenger.

Challenger Alice decides, at random, to use one of the two messages from adversary Bob. Alice encrypts the chosen message using her own public key $T_a(x)$ and the random masking parameters specified in section 3.4, and sends the ciphertext back to Bob (note: Alice is able to decrypt the message using her private key). Adversary Bob now tries to guess which of the two messages Alice has selected for encryption. Adversary Bob wins if he has more than a negligible advantage in guessing the correct choice the challenger has made.

The proposed ephemeral key exchange is IND-CPA secure. With every new key exchange, the adversary can learn nothing about the challenger's choice because of the strong pseudorandom behavior of the encryption algorithm. The adversary is consequently deprived of any advantage.

The IND-CCA2 Scheme

The game for adaptive chosen ciphertext attacks (IND-CCA2) differs from the first game by allowing adversary Bob to send further messages after having received the encrypted message, except the encrypted message itself, Alice answering immediately with the encrypted message. Neither x nor a are changed during this communication, the masking parameters described in section 3.4 are however chosen randomly for every message. Again Bob has to guess which of the two messages was encrypted in the first place.

The proposed ephemeral key exchange is IND-CCA2 secure because the masking is pure pseudorandom and cannot be removed without knowledge of the secret parameter. The adversary is consequently deprived of any advantage because different plaintexts will result in ciphertexts with pseudo-random nature.

The Key Generation Oracle Scheme (Side Channel Attack)

Deviating from the above schemes, Alice has control only of the static secret key r, and has to return $T_r(x)$ for every submitted value x. An attacker may provide a huge number of carefully chosen values for x to a token device (for example) and save the answers.[3] Is the attacker capable of gathering enough information in order to recover the secret key r ?

3 We already limited the precision of the public parameter x , but the public key of an adversary may appear with full floating point precision. The resulting shared secret can however not be seen by the adversary, but in multi-threaded systems there may be a central „oracle“ that does not know the origin of the supplied value, and therefore has to answer whenever asked to do so.

Attacker Eve may follow the following strategies:

(1) Eve can try to find a root (or, less efficient, an extremum) of $T_r(x)$ by varying x in small portions by the nested interval method. Eve can easily find an interval containing only one root, and determine the root with the precision of the token‘s I/O through repeating this process. The nested inverval method generates one result bit per cycle, so Eve will need to run $O(n)$ cycles with n being the length of the mantissa in bit of the floating point numbers.

Having determined a root, Eve has to find a pair of integers k, r that solve

$$\frac{2*k-1}{2*r} = \frac{acos(x)}{\pi}$$

A working option is to develop the floating point number on the right by the continued fractions method described in section 2.3 to get some good estimations of possible pairs k, r. We experimentally followed this path by trying to find a secret of 30 digits only. The pecision of the floating point numbers was 360 decimal digits, which is four times the ratio of the floating point precision / secret digits of the standard algorithm implementation. In most cases the candidates don‘t even hit the magnitude of the values searched for, not to speak from extracting some digits systematically.

We expected these results because of the inevitable rounding errors present (ref. section 2.3). Although, from a mathematical point of view, $acos(x)/\pi$ has to be a rational number and this never happens in real computations.

(2) Eve may try to find values for x for which a corresponding k is easy to determine. If Eve arrives here, she can compute r without any problem. Small k, or values $k \approx r$ result in x approaching the forbidden $x = 1.0$ (e.g. $x = 0.9999999999123456789$). However, Eve can for inctance determine r exactly for $k = 1$ in a scan search, i.e. Eve searches the root next to $x = 1$. An alternative way to determine r for another easy to determine second number that might allow for exact determination, $k = r/2$, i.e. looking for the root next to $x = 0$, does not yield r because of rounding errors.

Clearly the distance between adjacent roots is a function of the secret parameter r. If Eve determines another root some distance away from $x = 1.0$, Eve has to count the roots in between $x = 1.0$ and the chosen value to get the exact k to perform the calculation. Choosing a wrong k results in a wrong value for r .The more x differs from +/-1.0, the more equally probable values for k exist:

D = 1.0-x	**$\mathbf{Log_{10}(k)}$**
10^{-120}	50
10^{-70}	75
10^{-20}	100

Table 3: Relationship between the vicinity of x from 1.0 and the number of equally probable k (stated in $\log_{10}(k)$). The table is valid for the 100 digits shared secret size.

Table 3 is best interpreted this way: the last line depicts the effect of only allowing x to come as close as 20 decimal digits to 1.0. The uncertainty of an adersary to guess k is in the order of Alice's secret parameter r then, i.e. Eve would have to count $O(10^{100})$ roots between the greatest allowed value for x and 1.0 to yield the exact value for k, and she can get no advantage from this attack. If the adversary sends values near 1.0, the token device should discard the input, and should return an error message. Because x will normally be a pseudorandom value (a common x or a public key $T_b(x)$, not a value chosen at will by the attacker), critical x approaching 1.0 by as close as $D = 10^{-20}$ in normal operation will appear with probability $w = 10^{-20}$. This is definitely acceptable for practical implementations.

(3) Due to the pseudo-periodic behavoir of the analytical function $T_r(x)$, small differences in x will return values from the same period. This may give the attacker the chance to estimate the local period length Δx of $T_r(x)$, and thus to estimate

$$\Delta x = \cos(\pi * \frac{2*k-1}{2*r}) - \cos(\pi * \frac{2*k+1}{2*r}) \quad .$$

Again, Eve has to find a pair k, r solving the equation. The advantage of this strategy is the possibility to derive m systematically coupled equations containing the same k, r by the m - fold effort to determine one distance:

$$\Delta x(m) = \cos(\pi * \frac{2*k-1}{2*r}) - \cos(\pi * \frac{2*(k+m)-1}{2*r})$$

Eve may follow the evalution path suggested in (1), or may try to narrow the interval for k (2) by some means to decrease the number of k/r candidates.

An algebraic solution for these formulas does not exist, so they have to be solved numerically. As there are two variables k, r to be determined, we arrive at the problems discussed above.

In the future, other possibilities to determine a pair k, r may be developed which may result in useful estimations. But at least all such attemps will heavily depend on the ability to inject parameters at very high precision. In order to prevent from this, we therefore introduce, as a further option, to limit the number of decimal digits to all RVB input- and output parameters to e.g. 200 decimal digits. As an example, a hardware token device simply ignores digits beyond this limit and returns output parameters that are already truncated. The size of the common parameter x anyways needs to be truncated (section 1.6). So this option additionally affects the public keys. For a secret size of 100 digits, a cutting procedure results in a limited precision of the determined root values:

Size of public value [decimal digits]	**Distance to root/extremum for** x
100	$> 10^{-3}$
150	$> 10^{-40}$
200	$> 10^{-90}$
250	$> 10^{-150}$
300	$> 10^{-200}$

Table 4: Effects on the accuracy of an adversary trying to hit a root or an extrema in a worst-case attack scenario

On the other hand, limiting the digit size of the public key also limits the number of equal digits in the shared secret:

Size of public value [decimal digits]	**Difference in Shared Secret**
100	$> 10^{-3}$
150	$< 10^{-35}$
200	$< 10^{-85}$
250	$<10^{-130}$
300	$< 10^{-130}$

Table 5: Reduction of size of shared key when adding reduction of the floating point accuracy for all public RVB parameters

Depending on the actually chosen limit, floating point accuracy has to be increased in order to arrive at the desired security level for the shared secret. As there seems to exist no effective attack at present, this optional measure should

not be mandatory for encryption applications like the ones proposed in chapter 3. Limiting the minimal distance from 1 of the input parameter x is however mandatory to prevent from this attack.

Note. This side channel attack applies to special applications with static secret parameters only. For ephemeral-only key exchanges, this worst case attack is anyways not applicable as the adversary has to deal with public and secret RVB parameters that are selected at random for each and every new key exchange. An adversary has no access to the secret parameters except for his own.

2.5 Quantum Computer Security

We assume that the reader is familiar with quantum computing principles:

- Today there are three types of quantum computers known:
  - The adiabatic quantum computer which is developed up to the lower level of the number of qubits necessary to attack cryptography, but is widely accepted to be not suitable for cryptanalytical attacks.
  - The algorithmic quantum computer which is THE candidate for attacking cryptography, but is developed up to only about 10 qubits today [9].
  - The bulk quantum computer being only a theoretical qc model as of today.
- Qubits are prepared in superpositioned states at the beginning of the computation, and are entangled during computation. Entanglement of all qubits that hold the results with those holding intermediate results must be maintained from the beginning of the computation to the end. Quantum computation doesn't allow for storing intermediate results, or computing only parts of the desired result and combining them later to the complete solution by other means.
- All operations must be fully reversible to allow all assistant qubits to settle to pure states before reading out the result because of entanglement of all qubits.
- Quantum states are not stable, rendering error correction to a central necessity for quantum computers [10]. Error correction for one qubit consumes 2 .. 8 further qubits, which carry no information about the actual state when used to perform a correction operation by a measurement. As long as noinformation is leaked, they may theoretically be appended during computation to increase the limited computation lifetime to a level that is necessary to solve the problem, but this also increases the number of operations, and qubits, and the complexity of the qubit infrastructure.
- It is necessary to bring qubits into direct contact with each other in order to allow for entanglement to take place. A quantum computer consisting of several thousand qubits must be able to selectively combine every qubit with every other qubit in entanglement distance, or to allow for other techniques to enable entanglement operations between distributed qubits by means of, for instance, chains of helper qubits, thus resulting in a high complexity of the qubit infrastructure for arranging the qubits of the quantum registers in a way that entanglement can take place.

Although the proposed algorithm operates over $\mathbb{Q}$, we start our investigation with a solution for T polynomials over finite modular fields with an n bit integer representation to gain an insight of the principle effort of a quantum computation. With recourse to the matrix exponentiation algorithm, the exponent variable m – the value that is searched for – is initialized with an appropriate superposition of possible states. Since the generation matrix $\mathbf{A}$ is well known (x is a publicly known value), all values of matrix elements in the powers can be hidden as precalculated constants in the quantum operations, and no qubit is necessary for storage. Theresult matrix $\mathbf{R}$ however is initialized with the identity matrix, updated in each powering step and is related by some means to the second public value y. Since all qubits are entangled, the comparing step reacts on the exponent register increasing the probability to measure a usable result in the end. After having executed one or more powering steps resulting in a new intermediate $\mathbf{R}$, the algorithm is rolled back which means that the new intermediate result is now used to erase the old one leaving the qubits in pure states, and thus just allowing to reuse them in the further calculation, or to allow a measurement of the final result with all informations concentrated in the result qubits. Further details of this rough abstract of the computation depends on the quantum algorithm that is actually used.

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

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

When executing the complete algorithm, the QC programmer may follow the following strategy: he calculates R_{k+1} and uses this matrix to erase R_k (= preparing the qubits of R_k in a pure $|0\rangle$ state) immediately by multiplying R_{k+1} with A^{-1}, the inverse again hidden in the quantum operations (this is the standard procedure to save qubits, but an in-depth analysis of the possible strategies may come to another conclusion). Modular squaring operations are available at a cost of $O(n^3)$ operations and n matrices need to be calculated, thus the amount of operations increases to the order of magnitude of $O(n^4)$.

As can be seen from the formula, multiplication and addition cannot be done “in situ”. The carrying out of an operation is available at the cost of two temporary registers with 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. This increases the number of registers to implement to 1+4+2+8=15. In order to operate on 2,048 bit values, at least 30,720 qubits are needed (RSA factorization consumes only about 6,100 qubits), and the operations to be carried out sums up to a value that is greater than 2048 cycles * 12 squaring operations per cycle on average * $90*\tilde{n}^3 \approx 2*10^{16}$. [4]

Neither quantum control nor quantum correction is so far included in this calculation, which is however inevitable to carry out the quantum computation because pure quantum systems used as the base for the operations described above only have a lifetime of a few seconds at best. In the literature several models are discussed, adding up from individual control qubits for every qubit used in the calculation to virtual qubits in surface code models. In theory it seems to be possible to construct a quantum computer with the capability to break 2,048 bit RSA, which, due to quantum control and correction, requires several giga qubits, and may be long term stable, but increases the computation time from a few seconds in the ideal case to a couple of weeks at best [1] [19].

Assuming a QC which is capable to operate in the described area on finite fields can be built, and is still a threat to classical public key systems from a practical point of view. We now have to cope with the framework condition of our proposal, namely using $\mathbb{Q}$ instead of modular fields, and using floating point numbers with rounding operations in the implementation of the algorithm. Quantum algorithms have to be strictly invertible, but due to rounding operations, the operands can not be reconstructed from the result due to loss of information.

To show the consequences, we regard a typical operation sequence of the kind we described above in more detail:

$$\boldsymbol{U}|x\rangle|0\rangle|0\rangle|0\rangle \rightarrow |x\rangle|y\rangle|Z\rangle|0\rangle$$

$$\boldsymbol{CNOT}|x\rangle|y\rangle|Z\rangle|0\rangle \rightarrow |x\rangle|y\rangle|Z\rangle|y\rangle$$

$$\boldsymbol{U^{-1}}|x\rangle|y\rangle|Z\rangle|y\rangle \rightarrow |x\rangle|0\rangle|0\rangle|y\rangle$$

$$\boldsymbol{V^{-1}}|x\rangle|0\rangle|0\rangle|y\rangle \rightarrow |x\rangle|x\rangle|Z\rangle|y\rangle$$

$$\boldsymbol{CNOT}|x\rangle|x\rangle|Z\rangle|y\rangle \rightarrow |0\rangle|x\rangle|Z\rangle|y\rangle$$

$$\boldsymbol{V}|0\rangle|x\rangle|Z\rangle|y\rangle \rightarrow |0\rangle|0\rangle|0\rangle|y\rangle$$

Here, from a starting state $|x\rangle$ and three empty registers $|0\rangle$, a final state $|y\rangle$ is produced. All registers used in between are in the same pure states in the beginning as well as in the end. The first three rounds are used to produce $|y\rangle$ and $|x\rangle$, the next three rounds are used to erase $|x\rangle$. This approach is easily possible in modular operations over finite fields, but using something like rounding operations (for instance a CNOT operation on only a part of the complete register) would prevent from the possibility to roll back the algorithm later on. But rolling back is inevitable because the result register has to be the only one not in a known pure state when the measurement takes place. The best result that can be obtained is producing an end state $|x\rangle|y\rangle$ for each round of the algorithm.

A rough estimation for a secret of 100 decimal digits (~ 340 bit) leads to the following number of required qubits: the floating point precision is 3 times the secret magnitude, resulting in ~ 1,000 qubit per value. For each round of the exponentiation algorithm, two matrices with 4 elements each demand ~8,000 qubit as input, and ~16,000 qubit to

4 The proposed qubit numbers may vary according to some improvements of the base algorithms, but the increased requirement nevertheless remains.

generate the results. At the end, ~8,000 qubit are used as the input for the next round, the other ~8,000 qubit have to be maintained until they are erased in the rollback operation after the termination of the whole algorithm. Since there are ~340 rounds in total (one round for each bit of the secret), this sums up to 2.82 * 10^6 qubits, error control and correction yet not considered. As this is the lower bound of the key length, and the number of qubits will increase at the power of two if the secret bit size is increased, the number of qubits alone will represent „a big nut“ for quantum computer researchers to construct computers of such magnitude, or to develop new algorithms decreasing the number of qubits.

Assuming these problems can be solved, we have to look at the quantum algorithms in detail. Today, only two principle quantum algorithms might be applicable:

- Grover’s algorithm [12] looking directly for a solution of the problem, and
- Shor’s algorithm [7, 8, 14] looking for periodical events that can be used to solve the problem.

Grover’s algorithm is the quantum computational equivalent of a classical brute force attack, and is known to cut the complexity of the problem from $O(n)$ to $O(\sqrt{n})$, meaning there are $O(\sqrt{n})$ consecutive operations without any interruption necessary on the QC to yield a useful result. Grover’s algorithm is - without any doubt - a quantum algorithm that can theoretically be used to attack the proposed encryption algorithm. But regardless of whether a QC with the required lifetime could be built, the amount of time using Grover’s algorithm to crack the problem increases exponentially with increasing bit count of the secrets. Grover’s algorithm is consequently not a risk for the proposed algorithm.

Shor’s algorithm is looking for periods in the exponential development which are present in modular arithmetics, and therefore can break RSA keys, and those of other modular crypto systems. As a result of the theorems in section 1.2 the RVB algorithm doesn’t have any periodic states in its working range. Consequently Shor’s algorithm cannot be applied for pure mathematical reasons. Grover’s algorithm is therefore the best algorithm available today.

It can be concluded that algorithms using quantum effects known today cannot outperform algorithms running on classical computers in order to successfully attack the proposed key negotiation algorithm.

Attack on algorithm	Number of qubits	Number of operations per try
Shor's algorithm on RSA, n = 2,048 bit	~ 3 n = 6,000	~ $30n^4$ = 5.3*10^{14}
Grover's algorithm on T polynomials over $\mathbb{Q}$, n = 1,024 Bit	~ 15 * n^2 = 15 * 1024^2 = 1.6 *10^7	~ 1.2 * 10^{15} * $\sqrt{2^{1,024}}$ ~ 10^{170}
Shor's algorithm on T polynomials, n = 1,024 Bit	Not applicable	Not applicable

Table 6: Comparison of QC attacks on RSA and the proposed quantum-resistant public key exchange excluding quantum control and quantum correction

3 Further applications of the RVB key exchange

3.1 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 large number of participants.

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

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

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:

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

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.

3.2 Certificates and Proof of Knowledge

The classic X.509 certificate scheme binds public and private parameters to the identity of the owner thus allowing for authentication in addition to encryption. The certificates are signed by certification authorities, and the recipient of a certificate has only to check the signature. Static signatures of this kind can however not be created with the proposed algorithm. If authentication is wanted, one possibility is to prove a certificate in a live communication with the CA using the private authentication scheme discussed below, and to store verified certificates on the user system to decrease network traffic with the CA.

3.3 El Gamal type encryption scheme

If Bob‘s public parameters $x,y=T_b(x)$ are verified by a certificate (or an arbitrary other protocol), Alice can use an El Gamal scheme to encrypt a message to Bob. Alice chooses a secret value a , calculates $R=T_a(x),S=T_a(y)$, and masks the message N by

$$Q=sha_3(S)\oplus N$$

The pair R,Q is sent to Bob who decrypts the message by

$$N=Q\oplus sha_3(T_b(R))$$

Of course only the significant digits of $S,T_b(R)$ can be used in this operation.

3.4 Private Authentication and private Signature

We present two protocols to arrive at a shared long term secret between Alice and Bob:

1. If Alice and Bob used a common public parameter x to set up their certificates with the key pairs $y_a=T_a(x)$, $y_b=T_b(x)$, they immediately arrive at a shared secret key $K=T_{a*b}(x)$ without further action.
2. If Alice and Bob used individual parameters x_a,x_b to set up the certificates $y_a=T_a(x_a)$, $y_b=T_b(x_b)$,

an initial protocol has to be used once to arrive at a shared key. They both choose random numbers R_a, R_b and send them El Gamal-encrypted to the respective partner, $K = R_a \oplus R_b$ being their long term shared secret. This procedure is necessary to force both partners to use their secret in the negotiation.

This key, however, cannot be used in communication because it is a constant key. On the other hand they can be sure that only the partner is in possession of the correct key.

In order to communicate, they agree on a random value R which can be publicly negotiated. The individual session key is built by computing

$$K_{sess} = sha_3(K \mid R)$$

The shared long term secret can also be used as a private signature in asynchronous communication with the message transmitted in clear. Sending a message M, Bob adds

$$S = sha_3(K \mid M)$$

as a signature. Alice knows that only Bob could have produced this MAC. But Alice cannot convince a third party that the value originates from Bob. Alice can even not convince anybody that K is a negotiated secret shared with Bob as long as Bob doesn't admits that this is the case.

3.5 Hidden negotiated Signature

In most cases the authentication of the sender of a message is sufficient. But there may be applications where the authorship of the document or the document itself has to be proved. A provable document is defined by using the terms introduced previously:

$$D = \{Q_a, R_a, Q_b, R_b, AES(K, M)\}$$

with M being the message in the clear, K being the key to encipher M, and the pairs Q_x, R_x being the El Gamal enciphered K using the certificates of Alice and Bob. Alice, and Bob may verify by an appropriate protocol that all values are valid, but because the mathematics of RVB don't feature any invertible parts, either Alice, or Bob can produce such documents without involving their partner, so D alone cannot be proven without further assistance.

We thus introduce a trusted third party as a bookkeeper. Both Alice and Bob must be registered as verified users by their certificates (which may be master certificates, not those used for the encryption of the message). All messages between Alice, Bob, and the bookkeeper are handled by the authenticated scheme above. As a signature, Alice and Bob send

$$H_a = HMAC(S_a, D) \quad , \quad H_b = HMAC(S_b, D)$$

to the bookkeeper using the secret values that were used to generate the El Gamal pairs Q_x, R_x. In order to assure the bookkeeper that they are referring to the same D, they agree on a further random value T and send both

$$H = HMAC(T, D)$$

to the bookkeeper. Because Alice and Bob will have processed D before, and found it a correct document, the bookkeeper may assume that they both want to sign the message, and stores $\{H, H_a, H_b\}$ as the signature.

Note: the bookkeeper doesn't have any knowledge about the document D, because he doesn't know T. He can't even link a given document D' to any triple $\{H, H_a, H_b\}$ in his database, and it is possible for Alice and Bob to agree on some faked signature values for camouflage. The bookkeeper has only the value H to link H_a, H_b he got individually from Alice, and Bob.

In case the authorship of a document has to be proven, the following procedure is executed: if Alice wants to prove the signature, she presents the tuple $\{M, S_a, T, Q_a, R_a, Q_b, R_b\}$ and the certificate she used to a judge. The judge can thus decrypt K from Q_a, R_a, encrypt M, build D, and calculate the signature values H_a, H. He can now identify the triple $\{H, H_a, H_b\}$ in the bookkeeper's database. Because the bookkeeper avows for the participation of Bob in the protocol, the authorship of Alice, or Bob, or both depending on the message itself is proved.

Note: the judge is however not able to check Q_b, R_b, H_b, but there is no necessity to do so because B has delivered H before, a value which he could only provide if he is the author, or the receiver of the message m. He was identified during this procedure by the bookkeeper through his master certificate in a private authentication scheme.

Alice was not able to provide these values alone. Bob is anyhow convicted – if he cooperates or not.

Assume now Bob is cheating and presents a tuple $\{M', S'_b, T', Q'_a, R'_a, Q'_b, R'_b\}$ to the judge, claiming Alice being the author, or receiver of M. The judge now can proceed as before. But to convince the judge, Bob has to choose his tuple so that at least

$$H = HMAC(T, D) = HMAC(T', D')$$,

for some document D registered by the bookkeeper which can only be the case if

$$Bitstring(T+D) = Bitstring(T'+D')$$

This may be done by shifting the boundaries, i.e. taking a longer T', and a shorter D'. But since D' comprised $AES(K', M')$, and only M' is provided to the judge, he can do so only if he has broken the AES encryption algorithm, or the secure hash algorithm. As a matter of fact, Bob cannot cheat.

It is important to remark that the signature certificate of Alice cannot be used in the future because private key has been disclosed to the judge. But this signing certificate neither has to be the one used to authenticate Alice during the communication with the bookkeeper, nor the certificate she used to communicate with Bob. Both can agree on individual signing certificates for each document using the private authentication scheme. By registering a document at the bookkeeper, Bob has agreed to Alice's signing certificate implicitly, and no further proof is necessary.
Although Alice's signing certificate cannot be used in the future because private key has been disclosed to the judge, elder signatures produced with the same certificate are not concerned because nobody can link S_a to $\{H, H_a, H_b\}$ in the bookkeeper's database, except for Alice and Bob. Even if enough information is leaked that an attacker Eve is able to present a valid tuple for another triple to the judge, she cannot claim to be the author of the presented document, because the secret value S_a is already linked to Alice.

3.6 Classiscal Static Signature[5]

Bob generates a certificate that contains a public key consisting of two rational numbers $x, y = T_r(x)$ with r representing Bob's secret key. r consists of two factors $r = r_1 * r_2$ with r_1 being a large random number,

$$r_2 = \prod_{k=1}^{n_p} p_k^{e_k}$$

being a number composed of a set $\{p_1, p_2, ... p_n\}$ of n_p primes with completely known exponent vector $\boldsymbol{e} = (e_1, ... e_{n_p})$.

In order to sign a message N, Bob computes the hash value $h = Hash(N)$. r_2 has to be large compared with h . Using $\boldsymbol{e}$, Bob selects a number h' with known exponent vector $\boldsymbol{e}'$: $\forall e'_k : e'_k \le e_k$ which is near h . Thus $h' \mid r_2, r$. Bob publishes the pair $s = h - h'$, $z = T_{r/h'}(x)$ as the signature.

Alice verifies the signature by calculating $y = T_{h+s}(z) = T_{h+s}(T_{r/(h+s)}(x)) = T_r(x)$.

By collecting a number of signatures, attacker Eve may learn about the composition of r_2 . But as Eve doesn't learn anything about r_1 , she cannot commit fraud because she is not able to compute z .

5 A signing algorithm is besides of KEM another target of the NIST PQC project. T Polynomials are generally suitable for both tasks. The actual algorithm is however still under development.

The divisor grid of r_2 has $M=\prod_{k=1}^{n_p}(e_k+1)$ members which is a small number compared with r_2. As a consequence, only a limited number b_s of the bits of h at the MSB side can be secured by a signature pair.

Knowing a signature pair, attacker Eve may try to find another hash value $h'\approx h$ within the range of the unsecured lower bits by brute force, and publish another pair h', $s'=(h+s)-h'$ which looks like a valid signature as well securing b_s bits too.

In order to yield a higher security level, Bob publishes an n-tuple of signature pairs $\left((s_1,z_1),(s_2,z_2),...(s_L,z_L)\right)$ securing L hash values generated by the procedure

$$h_k=hash(N \vee xx \vee s_{k-1})$$

with $s_0=0$ and xx being a binary number set to 0 and incremented by 1 until $h_k\geq 0x1000...000$. Because all signed values are interconnected by the procedure, but hash values of different inputs look like independent random numbers and therefore yield approximations with no interdependence, the security will approximate $\approx L*b_s$. If L is chosen such that $L*b_s>hashsize$, the signature may become lengthy but ultimately yields the full security of the hash function.

In the construction of the algorithm the following attack scenario is considered: If Eve is in possession of a signature pair h_a, $T_{r/h_a}(x)=z_a$ and wants to compute a signature pair for a hash value h_b with property $h_b=h_a/K$, she may compute z_b without knowing the secret through

$$T_{r/h_b}(x)=T_{(r/h_a)*K}(x)=T_K(T_{r/h_a}(x))=T_K(z_a)=z_b$$

Any signing algorithm therefore must avoid to generate signature pairs of different magnitude.

Bob may construct r_2 by arbitrary means. Given h, finding a h' with the given set $\left((p_k,e_k)\right)$ is a typical knapsack problem which can only be solved by some skillfully designed search algorithm. Preliminary results show that the sieve for an approximate value for a 256 bit sha3-hash value can be executed within a period of time that is acceptable for practical applications.

3.7 Group Secrets

In some applications 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

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

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

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

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 cheat and share their secret, the easier it is to mount an insider attack to unveil the secret s.

3.8 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

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

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

$$z_i \;,\; y_i=P_m(z_i)(mod\ p) \;,\; 1<i<n$$

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 by 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 are then erased from memory. For the next reconstruction of the secret, the whole process has to be started again.

It should be noted that modular arithmetics are used in this protocol in spite of the intrinsic vulnerability to QC attacks. In this scheme this although does not provide a possibility for an attack. All modular parameters are known to all peers, but this doesn't allow to reconstruct the polynomial as long as the peers don't share their values.

3.9 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. $(x, y=T_{s_i \cdot s'_i}(x))$ 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:

$$y=T_{s_i}(T_{s'_i}(x))$$
$$y_{alice}=T_{alice}(x)$$
$$SECRET=T_{s_i}(T_{s'_i}(y_{alice}))=T_{alice}(y)$$

The principle of numerics guarantees that the significant digits L are identical, whoever of the peer generates $T_{s \cdot alice}(x)$.

The region of lost digits by rounding operations however depends on the functions involved, and the consequence is that

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

The rounding region can therefore be used as a fingerprint of the acting peer. In case of a disclosure, the acting peer himself, or at least the other N-1 group members can verify who the actor was under the precondition that the values are derived honestly, and the complete y values comprising the unused digits in the secret negotiation are stored in the protocol. 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.

4 Summary

4.1 General

We have introduced a very fast public key negotiation algorithm using less bits in the key exchange protocol than comparable classic public key schemes of similar security levels, and being also faster if the same bit size representation is used because there are less operations necessary. The key exchange can even be directly integrated in the audio RTP stream of popular VoIP applications. The principle is in fact not new [3, 4, 5, 6], but the authors previously working on such schemes were not sufficiently aware of the power of numerics.

We have also shown that the algorithm is resistant to quantum computer attacks. The proof is purely mathematical and therefore there is no room for the argument of „future improvements in QC theory“ to break the algorithm.

The algorithm can even be used to create a static public signature scheme.

4.2 Algorithm summary

Secret Key Length	100 decimal digits = 332 bit, as well as 200 (=664 bit) and 300 decimal digits (=996 bit)
Other Secret Key Lengths	Arbitrary
Valid Common Values	$-1<x<1$, $x \notin \{\pm 0.5, 0\}$
Common Value Length	Arbitrary, but not exceeding the number of digits of the secret keys
Common Value Selection	Arbitrary for every key exchange – preferrably at random. A common value may however be fixed by a standard for use by an arbitrary number of users to simplify extended applications (see section 3)
Floating Point Precision	3 times the secret key length recommended
Public Key Length	Floating point precision, optionally reduced
Shared Secret Length	Secret key length, optionally reduced
Public Key Negotiation , El Gamal Encryption, etc	Equal secret key length recommended.
Protocol Parameters	A protocol using the algorithm for encryption must provide the following parameters: common value, public key, secret key size, public key size, shared secret size
Exceptions	The protocol must provide control and error messages for the shared key (see section 1.3, string representation; although not very likely, incompatible states may arise)
Execution speed of the reference implementation for the entire key exchange (Alice AND Bob) on an Intel i7-6700K quad core microprocessor operating at 4.00 GHz. 64 bit microcode (NIST PQC reference platform)	In 100 decimal digit mode: 6 ms without OpenMP (7 ms with „acceleration“ by OpenMP). In 200 decimal digit mode: 18 ms with OpenMP (22 ms without OpenMP). In 300 decimal digit mode: 35 ms with OpenMP (59 ms without OpenMP). If OpenMP is to be used, the numeric libraries must to be compatible with OpenMP, which needs to be checked.
RAM consumption of the reference implementation for the NIST PQC reference platform	In 100 decimal digit mode: 7 kByte. In 200 decimal digit mode: 11 kByte. In 300 decimal digit mode: 16 kByte.

4.3 Security Summary

Based on the values stated in section 4.2:

Primary Secret Range	332 bit random integer for setting to 100 decimal digits, 664 bit random integer for setting to 200 decimal digits and 996 bit random integer for setting to 300 decimal digits.
Shared Secret Range	332 bit pseudo-random floating point value for setting to 100 decimal digits, 664 bit pseudo-random floating point value for setting to 200 decimal digits and 996 bit pseudo-random floating point value for setting to 300 decimal digits.
Security when used in conjunction with a symmetric cipher	256 bit strength (e.g. for use with AES256) for 100 decimal digits (approx. 332 bit), approx. 505 bit strength for 200 decimal digits and approx. 743 bit strength for 200 decimal digits. Considering the results in section 2.2, and in order to be conservative in security estimations, we assume that a 332 bit (100 decimal digit) shared key features a security equivalent to a symmetric cipher key of 256 bit.
Further Increase of Security	The security level may be adjusted by • increased secret size (floating point precision has to be adjusted accordingly) • public key size limitation depending on the characteristics of attack to prevent from (shared secret size has to be adjusted accordingly)
Modification Options	In order to counter the most extreme types of attacks mentioned in section 2.4, public parameters must be kept away from roots and extrema. A repeated key negotiation with different values should be enabled by the protocol. Furthermore the limitation of the digits size of all public values to values lower than the floating point precision is possible. The executing device must cut the length to the values given in the protocol parameters prior to any computation. This limiting of parameter size will have a negative impact on the shared secret size.
Immunity from Chosen Plaintext / Adaptive Chosen Plaintext Attacks	IND-CPA and IND-CCA2 The way RVB is applied is identical with Diffie Hellman / El Gamal. We have no evidence that an attacker can profit from sampling up to 2^{64} input/output pairs and using an „oracle“.
Quantum Computer Security	Shor‘s algorithm: Not applicable due to mathematical properties. Grover‘s algorithm: algorithmic attack may be implemented
Qubit Demand in QC Attacks	$O(pre^2)$ due to floating point arithmetics, *pre* being the floating point precision present in the classical calculation
Quantum Gates	$O(pre^3 * e^{pre/2})$ for full execution of Grover‘s algorithm
Quantum Computer Security	Even under NIST‘s assumptions of the QC development path we think that our basic RVB implementation is a „tough nut“ for a QC to crack. Sizes of secret parameters may be increased arbitrarily. Due to the quadratic increase in qubit / gate count, we assume that the algorithm can outrun practical QC easily for a long time. It should be kept in mind that Grover‘s algorithm execution time increases exponentially in the number of bits!

References:

[1] B. Lekitsch, S. Weidt, A. Fowler, K. Molmer, S. Devitt, C. Wunderlich, W. Hensinger, Blueprint for a microwave trapped ion quantum computer, 2017, http://www.physik.uni-siegen.de/quantenoptik/forschung/publikationen/publis/sciadv_e1601540.pdf

[2] E. Weisstein, Chebyshev Polynomial of the First Kind, from MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/ChebyshevPolynomialoftheFirstKind.html

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

[4] Garry J. Tee, Permutable Polynomials and Rational Functions, 2011, http://nzjm.math.auckland.ac.nz/images/3/31/Permutable_Polynomials_and_Rational_Functions.pdf

[5] P. Bergamo, P. D'Arco, A. De Santis, L. Kocarev. Security of Public Key Cryptosystems based on Chebyshev Polynomials, 2004, http://www.di.unisa.it/~paodar/preprint/pdf/web_chaos.pdf

[6] L. Kocarev, Z. Tasev, P. Amato, Rizzotto, Encryption process employing chaotic maps and digital signature process, 2005, US Patent 6,892,940 D2 (Foreign Application Priority: Apr. 7, 2003, EP 03425219)

[7] Shor's Factoring Algorithm, Notes from Lecture 9 of Berkeley CS 294-2, 4 Oct 2004, http://www.cs.berkeley.edu/~vazirani/f04quantum/notes/lec9.ps

[8] IBM's Test-Tube Quantum Computer Makes History, Press release, 19 Dec 2001, http://www-03.ibm.com/press/us/en/pressrelease/965.wss

[9] 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

[10] IBM Scientists Achieve Critical Steps to Building First Practical Quantum Computer, Press release, 29 Apr 2015, http://www-03.ibm.com/press/us/en/pressrelease/46725.wss

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

[12] L. K. Grover, A fast quantum mechanical algorithm for database search, 1996, http://arxiv.org/pdf/quant-ph/9605043v3.pdf

[13] G. Marsaglia, Website: The Marsaglia Random Number CDROM including the Diehard Battery of Tests of Randomness, http://www.stat.fsu.edu/pub/diehard/

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

[15] BSI TR-02102-1, 2017, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile

[16] G. Brands, C.B. Roellgen, K.U. Vogel, QRKE: Quantum-Resistant Public Key Exchange, 2015,

https://www.researchgate.net/publication/282286331_QRKE_Quantum-Resistant_Public_Key_Exchange

[17] G. Brands, C.B. Roellgen, K.U. Vogel, QRKE: Extensions, 2015, https://www.researchgate.net/publication/283213947_QRKE_Extensions

[18] G. Brands, C.B. Roellgen, K.U. Vogel, QRKE: Resistance to Attacks using the Inverse of the Cosine Representation of Chebyshev Polynomials, 2016, https://www.researchgate.net/publication/291945494_QRKE_Resistance_to_Attacks_using_the_Inverse_of_the_Cosine_Representation_of_Chebyshev_Polynomials

[19] G. Fowler, Matteo Mariantoni, John M. Martinis and Andrew N. Cleland: Surface codes: Towards practical large-scale quantum computation, 2012, https://arxiv.org/abs/1208.0928