Ramstake

KEM Proposal for NIST PQC Project

September 29, 2017

cryptosystem name	ramstake
principal submitter	Alan Szepieniec imec-COSIC KU Leuven alan.szepieniec@esat.kuleuven.be tel. +3216321953 Kasteelpark Arenberg 10 bus 2452 3001 Heverlee Belgium
auxiliary submitters	-
inventors / developers	same as principal submitter; relevant prior work is credited as appropriate
owner	same as principal submitter
backup contact info	alan.szepieniec@gmail.com
signature	

Contents

1. Introduction

The long-term security of confidential communication channels relies on their capacity to resist attacks by quantum computers. To this end, NIST envisions a transition away from public key cryptosystems that are known to fail in this scenario, and towards the so-called *post-quantum* cryptosystems. One of the functionalities in need of a post-quantum solution that is essential for securing online communication is *ephemeral key exchange*. This protocol enables two parties to agree on a shared secret key at a cost so benign as to allow erasure of all other secret key material immediately after execution as an additional security measure. In the case where the order of the messages need not be interchangeable, this functionality is beautifully captured by the *key encapsulation mechanism* (KEM) formalism of Cramer and Shoup [5].

The desirable properties of a post-quantum KEM are obvious upon consideration. It should be fast and it should generate short messages, not require too much memory and be implementable on a small area or in a few lines of code. It should inspire confidence by relying on long-standing hard problems or possibly even advertising a proof of security. However, this design document is predicated on the greater importance of a property not included in the previous enumeration: *simplicity*. The requirement for advanced degrees in mathematics on the part of the implementers presents a giant obstacle to mass adoption, whereas no such obstacle exists for mathematically straightforward schemes. More importantly, complexity has the potential to hide flaws and insecurities as they can only be exposed by experts in field. In contrast, a public key scheme that is accessible to a larger audience is open to scrutiny from that same larger audience, and should therefore engender a greater confidence than a scheme that only a few experts were not able to break.

This document presents Ramstake, a post-quantum key encapsulation mechanism that excels in this category of simplicity. Aside from the well-established tools of hash functions, pseudorandom number generators, and error-correcting codes, Ramstake requires only high school mathematics. In terms of the categories of message size and speed, Ramstake does not outperform its competition but is still competitive with messages of less than four kilobytes generated in a handful of milliseconds on a regular desktop computer. Ramstake's biggest flaw is that its security relies on a relatively new and under-studied hard problem. It requires several years of attention attention from the larger cryptographic community before it can inspire confidence.

Mathematical Innovation. In a nutshell, this hard problem requires finding *short*-and-*sparse* solutions to linear equations modulo a large prime. If p is the prime modulus, then the solutions (x_1, x_2) have size on the order of $x_1 \sim x_2 \sim p^{3/4}$. Moreover, the balanced ternary expansion of x_1 and x_2 (*i.e.* using coefficients $1, 0$ and -1) consists overwhelmingly of zeros. Specifically, these integers can be described as

$$\pm\, x_i = (-1)^{s_0} 2^{e_0} + \sum_{j=1}^{\varsigma} (-1)^{s_j} 2^{e_j} \quad , \tag{1}$$

where the s_j are random integers drawn from $\{0, 1\}$ and the e_j from $\{0, \ldots, \frac{3}{4}\pi - 1\}$ except for $e_0 \in \{\frac{1}{2}\pi, \ldots, \frac{3}{4}\pi - 1\}$. The $\pm$ sign indicates that x_i must be positive. The *sparsity* parameter ς determines the number of terms. Ramstake's analogue of the discrete logarithm problem requires finding x_1 and x_2 of this form from G and $H = x_1 G + x_2 \bmod p$.

An important ingredient is the choice of the modulus p, which must be a pseudo-Mersenne prime. In other words, p can be described as $p = 2^{\pi} - t_{\pi}$, where $t_{\pi} \sim \log p$ is the smallest natural number such that p is prime. The parameter $\pi = |p|$ indicates the number of bits in the binary expansion of p and therefore also the minimal number of bits needed to represent any integer between 0 and p.

At this points, all tools for a noisy Diffie-Hellman protocol [2, 3] are in place. Alice and Bob agree on a random integer G between 0 and p. Alice chooses short-and-sparse integers x_1 and x_2 and sends $H = x_1 G + x_2 \bmod p$ to Bob. Bob chooses short-and-sparse integers y_1 and y_2 and sends $F = y_1 G + y_2 \bmod p$ to Alice. Alice computes

$S_a = x_1 F \bmod p$ and Bob computes $S_b = y_1 G \bmod p$ and both integers approximate $S = x_1 y_1 G \bmod p$ in the following sense: the differences $S_a - S = x_1 y_2 \bmod p$ and $S_b = y_1 x_2 \bmod p$ have a sparse balanced ternary expansion in the first $\pi/2 - |t_\pi|$ components. Therefore the first $\pi/2 - |t_\pi|$ bits in the binary expansion of S_a and S_b agree in most places. Alice and Bob have thus established a shared noisy secret stream of data, or since it will be used as a one-time pad, a *shared noisy one-time pad* (SNOTP, "snow-tipi").

From SNOTP to KEM. There are various constructions in the literature for obtaining KEMs from SNOTPs, each different in its own subtle way. The next couple of paragraphs give a high-level description of a generic transformation targeting IND-CCA2 security, which is inspired by the "encryption-based approach" of NewHope-Simple [4]. This construction makes abstraction of the underlying mathematics.

The encapsulation algorithm is a deterministic algorithm taking a fixed-length random seed s as an explicit argument. If more randomness is needed than is contained in this seed, it is generated from a cryptographically secure pseudorandom number generator (CSPRNG). The algorithm outputs a ciphertext c and a symmetric key k.

The encapsulation algorithm uses an error-correcting code such as Reed-Solomon or BCH to encode the seed s into a larger bitstring. Then the ciphertext c consists of two parts: 1) a contribution to the noisy Diffie-Hellman protocol; and 2) the encoding of the seed but one-time-padded with the encapsulator's view of the SNOTP. The decapsulation algorithm computes its own view of the SNOTP using the Diffie-Hellman contribution and undoes the one-time pad to obtain the encoding up to some errors. Under certain conditions, the error-correcting code is capable of retrieving the original seed s from this noisy codeword. At this point, the decapsulation algorithm runs the encapsulation algorithm with the exact same arguments, thus guaranteeing that the produced symmetric key k is the same for both parties. Robust IND-CCA2 security comes from the fact that the decapsulator can compare bit by bit the received ciphertext against the one that was recreated from the transmitted seed.

2. Specification

2.1. Parameters

The description of the scheme references the following parameters without reference to their value. Concrete values are given in Section 2.4.

- p — the prime modulus;
- π — the number of bits in the binary expansion of p;
- t_π — the difference between 2^π and p;
- ς — the sparsity, which determines the number of nonzero coefficients in the short-and-sparse secrets;

- ν — the number of codewords to encode the transmitted seed into;
- n — the length of a single codeword (in number of bytes);
- κ — the targeted security level (in $\log_2$ of classical operations);
- λ — the length of seed values (in number of bits);
- χ — the length of the symmetric key (in number of bits).

2.2. Tools

2.2.1. Error-Correcting Codes

Ramstake relies on Reed-Solomon codes over GF(2^8) with designed distance $\delta = 224$ and dimension $k = 32$. Codewords are $n = 255$ field elements long and if there are 111 or fewer errors they can be corrected. The following subroutines are used abstractly:

- rs_encode_multiple takes a string of $8k = 256$ bits and outputs a sequence of $8n\nu$ bits that represents ν times the Reed-Solomon encoding of the input.
- rs_decode_multiple takes a string of $8n\nu$ bits representing ν noisy codewords and tries to decode all of them. If there are multiple candidate received messages, the candidate that leads to a multiple encoding with the smallest number of bit errors is chosen and returned. If there is only one candidate received message, it is returned. If none of the codewords is decodable, this routine returns the error symbol $\perp$.

This abstract interface suffices for the description of the KEM. Moreover, any concrete instantiation can be exchanged for any other instantiation that adheres to the same interface, or that modifies the interface slightly to retain compatibility.

2.2.2. CSPRNG

Ramstake relies on a CSPRNG that stores a state and updates it as pseudorandomness is generated. The following member methods of a CSPRNG object `rng` hide this storage, access and updating of an internal state:

- `rng`.init takes no input and initializes the state to fixed starting point, possibly all zeros.
- `rng`.seed takes an arbitrary length input and uses it to seed the random number generator.
- `rng.generate` takes an integer ℓ and outputs a pseudorandom bit string of length ℓ.
- `rng.generate_ulong` takes no input and generates a bitstring of 64 bits, to be interpreted as an integer.

This abstract interface suffices for the description of the KEM. Like in the case of the Reed-Solomon codec any concrete instantiation can be exchanged for any other instantiation that adheres to the same interface.

2.3. Description

2.3.1. Data Structures

Ramstake uses five data structures: a random seed, a secret key, a public key, a ciphertext, and a symmetric key. Random seeds are bitstrings of length λ, whereas symmetric keys are bitstrings of length χ. The other three data structures are more involved.

Secret key. A secret key consists of the following items:

- `seed` — a random seed which fully determines the rest of the secret key in addition to the public key;
- a, b — short-and-sparse integers, represented by π bits each.

Public key. A public key consists of the following items:

- `g_seed` – a random seed which is used to generate the random integer G;
- C — integer between 0 and p which represents a noisy Diffie-Hellman contribution. This value satisfies $C = aG + b \bmod p$.

Ciphertext. A ciphertext consists of the following items:

- D — integer between 0 and p which represents a noisy Diffie-Hellman contribution; this value satisfies $D = a'G + b' \bmod p$ where a', b' are secret short-and-sparse integers sampled by the encapsulator;
- `seedenc` — string of $8n\nu$ bits; this value is the bitwise xor of the binary expansion of $\lfloor S \cdot 2^{8n\nu-\pi} \rfloor$ and $\mathsf{rs\_encode\_multiple}(s)$, where s is the random seed that is the argument to the encapsulation algorithm, and where S is the encapsulator's view of the SNOTP: $S = a'(aG + b) \bmod p$.

Using this same notation, the symmetric key $k \in \{0,1\}^\chi$ satisfies $k = \mathsf{H}(a'aG + a'b + b' \bmod p)$, where H is the SHA3-256 hash function with output truncated to χ bits.

All integers are serialized as length-π bit strings with most significant bit first. The three complex objects are serialized by appending the serializations of their component objects in the order as they are presented. No length information is necessary as the size of each object is a function of the parameters.

2.3.2. Algorithms

A KEM consists of three algorithms, KeyGen, Enc, and Dec. Pseudocode for Ramstake's three algorithms is presented in Algorithms 2, 3, and 4. In addition to that, KeyGen and Enc rely on a common subroutine called sample_integer which deterministically samples a short-and-sparse integer given the CSPRNG object, and which is described first in Algorithm 1.

algorithm sample_integer
input: `rng` — a CSPRNG object
output: $a \in \mathbb{Z}$ — a short-and-sparse integer

1: $u \leftarrow$ `rng`.generate_ulong()
2: $s \leftarrow u \bmod 2$
3: $e \leftarrow \lfloor \frac{u}{2} \rfloor \bmod \frac{1}{4}\pi$
4: $a \leftarrow (-1)^s \cdot 2^{e+\pi/2}$
5: **for** $i \in 1, \ldots, \varsigma$ **do:**
6: $\quad u \leftarrow$ `rng`.generate_ulong()
7: $\quad s \leftarrow u \bmod 2$
8: $\quad e \leftarrow \lfloor \frac{u}{2} \rfloor \bmod \frac{3}{4}\pi$
9: $\quad a \leftarrow a + (-1)^s 2^e$
10: **end**
11: **if** $a < 0$ **do:**
12: $\quad a \leftarrow -a$
13: **end**
14: **return** a

Algorithm 1: Procedure to sample a short-and-sparse integer from a CSPRNG.

algorithm KeyGen
input: $s \in \{0,1\}^\lambda$ — random seed
output: `sk` — secret key
`pk` – public key

▷ initialize objects
1: `rng1`, `rng2` ← new CSPRNG objects
2: `rng1`.init(), `rng2`.init()
3: `rng1`.seed(s)

▷ generate public key stuff
4: `g_seed` ← `rng1`.generate(λ)
5: `rng2`.seed(`g_seed`)
6: $G \leftarrow$ `rng2`.generate(π) ▷ interpret bitstring as integer

▷ generate secret key stuff
7: $a \leftarrow$ sample_integer(`rng1`)
8: $b \leftarrow$ sample_integer(`rng1`)

▷ generate more public key stuff
9: $C \leftarrow aG + b \bmod p$

10: **return** `sk` $= (s, a, b)$, `pk` $= (\texttt{g\_seed}, C)$

Algorithm 2: Generate a secret and public key pair.

algorithm Enc
input: $s \in \{0,1\}^\lambda$ — random seed
`pk` — public key
output: `ctxt` — ciphertext
$k \in \{0,1\}^\chi$ – symmetric key

▷ initialize objects
1: $\texttt{rng1}, \texttt{rng2} \leftarrow$ new CSPRNG objects
2: $\texttt{rng1}.\mathsf{init}(), \texttt{rng2}.\mathsf{init}()$
3: $\texttt{rng1}.\mathsf{seed}(s)$
4: $\texttt{rng2}.\mathsf{seed}(\texttt{pk.g\_seed})$
5: $G \leftarrow \texttt{rng2}.\mathsf{generate}(\pi)$ ▷ interpret bitstring as integer

▷ compute noisy Diffie-Hellman contribution
6: $a' \leftarrow \mathsf{sample\_integer}(\texttt{rng1})$
7: $b' \leftarrow \mathsf{sample\_integer}(\texttt{rng1})$
8: $D \leftarrow a'G + b' \bmod p$

▷ encode random seed and apply SNOTP
9: $S \leftarrow a' \, \texttt{pk.C} \bmod p$
10: $\texttt{seedenc} \leftarrow \lfloor S \cdot 2^{8n\nu-\pi} \rfloor$ ▷ bit expansion and pad with zeros as necessary
11: $\texttt{seedenc} \leftarrow \texttt{seedenc} \oplus \mathsf{rs\_encode\_multiple}(s)$ ▷ where $\oplus$ means bitwise xor

▷ compute symmetric key
12: $S \leftarrow S + b' \bmod p$
13: $k \leftarrow \mathsf{H}(S)$
14: **return** $\texttt{ctxt} = (D, \texttt{seedenc}), k$

Algorithm 3: Encapsulate: generate a ciphertext and a symmetric key.

algorithm Dec
input: `ctxt` — ciphertext
`sk` — secret key
output: k — symmetric key on success; otherwise $\perp^{(1)}$ indicating decoding error or $\perp^{(2)}$ indicating integrity error

▷ initialize objects
`rng1`, `rng2` ← new CSPRNG objects
`rng1`.init(), `rng2`.init()
`rng1`.seed(s)

▷ recreate public key object
`g_seed` ← `rng1`.generate(λ)
`rng2`.seed(`g_seed`)
$G \leftarrow$ `rng2`.generate(π) ▷ interpret bitstring as integer
$C \leftarrow$ `sk`.$a\, G +$ `sk`.$b \bmod p$

▷ obtain SNOTP and decode `seedenc`
$S' \leftarrow$ `sk`.a `ctxt`.$D \bmod p$
`str` $\leftarrow \lfloor S' \cdot 2^{8n\nu-\pi} \rfloor$ ▷ bit expansion, pad with zeros as necessary
`str` ← `str` $\oplus$ `ctxt.seedenc` ▷ bitwise xor
$s \leftarrow$ rs_decode_multiple(`str`)
if $s = \perp$ **do:**
    return $\perp^{(1)}$
end

▷ recreate and test ciphertext
`ctxt`$', k \leftarrow$ Enc(s, `pk` = (`g_seed`, C))
if `ctxt` $\neq$ `ctxt`$'$ **do:**
    return $\perp^{(2)}$
end

return k

Algorithm 4: Decapsulate: generate symmetric key and test validity of the given ciphertext.

2.4. Parameter Sets

2.4.1. Ramstake RS 22040

Ramstake RS 22040 is the main parameter set targeting the highest security level. It uses Reed-Solomon codes for error correction. The secret key is $\lambda/8 + 2\pi/8 = 5542$ bytes (5.41 kB); the public key is $\lambda/8 + \pi/8 = 2787$ bytes (2.72 kB); and the ciphertext is $\pi/8 + n\nu = 2755 + 1275 = 4030$ bytes (3.94 kB).

parameter	value
π	22040
t_π	2325
p	$2^{22040} - 2325$
ς	22
ν	5
n	255
κ	256
λ	256
χ	256

3. Performance

3.1. Failure Probability

There is a nonzero probability of decapsulation failure even without malicious activity. This event occurs when the two views of the SNOTP are too different, requiring the correction of too many errors. It is possible to find an exact expression for this probability. However, we opt for a more pragmatic approach.

Extensive tests with the Ramstake RS 22040 parameter set indicate that the mean number of symbol errors across a multiple-codeword of $n\nu = 1275$ symbols in total is $\mu = 469.44$ with standard deviation $\sigma = 20.419$. (The mean number of bit errors is $\mu = 952.84$ with standard deviation $\sigma = 51.909$.)

In order for a decoding failure to occur for the Ramstake RS 22040 parameter set, there must be at least 112 symbol errors in each received codeword to cause every decoding attempt to fail. Therefore there must be at least $112\nu = 560$ errors in total. Ignore the requirement that they must be spread out more or less equally, and model this distribution as normal. Then the probability of decoding error is at most

$$\Pr[\perp^{(1)}] \leq 1 - \Phi\left(\frac{560 - \mu}{\sigma}\right) \approx 4.6 \cdot 10^{-6} \ . \tag{2}$$

In the event of a decoding error the sender can use the same public key but encapsulate with a different random seed for an independent retrial.

3.2. Complexity

All three algorithms KeyGen, Enc and Dec are straight-line programs at the given level of abstraction. The subroutine sample_integer does involve a loop but as ς is

not intended to grow with the security parameter it might as well be considered unrolled. KeyGen requires two finite field operations, Enc requires four, and Dec requires 7.

This field operation presentation hides the complexity of large integer manipulation. Addition and subtraction are linear in π, but multiplication and modular reduction are both quadratic. Multiplication always involves at least one sparse integer, so this operation can be optimized to become a small number of additions and subtractions. Modular reduction involves a pseudo-Mersenne prime modulus so this can be optimized as well.

KeyGen and requires generating $\lambda/8 + \pi/8 + 16\varsigma$ bytes of pseudorandomness. Enc requires $\pi/8 + 16\varsigma$ bytes of pseudorandomness and one hash evaluation. Dec requires $\lambda/8 + \pi/4 + 32\varsigma$ bytes of pseudorandomness and one hash evaluation. KeyGen and Enc seed two random number generators with λ bits; Dec performs this operation four times.

In practice, the running time is dominated by Dec whose bottleneck is the Reed-Solomon decoder. Nevertheless, all algorithms are quite fast despite being poorly optimized. A test of 1000 key generations, encapsulations, and decapsulations, clocks in at 6.720 seconds on a system with four Intel i5-4590 CPU cores running at 3.30GHz.

4. Security

4.1. Hard Problems

Ramstake relies on the problem of finding short-and-sparse solutions to linear equations modulo a pseudo-Mersenne prime p, to be abbreviated by SSSLE. The formal problem statement is as follows.

Short-and-Sparse Solutions to Linear Equations (SSSLE) Problem. *Given:* A matrix $A \in \mathbb{F}_p^{m \times n}$ with $n \geq m$ and a target vector $\mathbf{b} \in \mathbb{F}_p^m$, both over a large prime field. *Task:* Find a vector $\mathbf{x} \in \mathbb{F}_q^n$ such that $A\mathbf{x} = \mathbf{b}$ and such that every component x_i of the solution can be described as

$$x_i = \sum_{j=1}^{\varsigma} (-1)^{s_j} 2^{e_j} \quad , \tag{3}$$

for some $s_j \in \{0, 1\}$ and $e_j \in \{0, \ldots, \frac{3}{4}\lceil \mathsf{log}_2\, p \rceil - 1\}$.

This problem is inspired by Aggarwal *et al.*'s Mersenne number cryptosystem [1], in which case the attacker's task is to decompose a given integer as a fraction of low-Hamming-weight integers modulo a large Mersenne number. It is simultaneously inspired by the Short Solutions to Nonlinear Equations (SSNE) problem [6], which merges the SIS and MQ problems. Indeed, the sparsity constraint is nonlinear, although establishing a full polynomial description of this infeasible.

Nevertheless, the SSSLE problem is the analogue of the discrete logarithm problem in Diffie-Hellman key agreement. What is needed for a proper security reduction is an analogue of the Decisional Diffie-Hellman problem, formally stated below.

Ramstake Diffie-Hellman (RDH) Problem. *Given:* Four integers (G, H, F, S) where $H = x_1 G + x_2 \bmod p$ and $F = y_1 G + y_2 \bmod p$ for some short-and-sparse (*i.e.*, satisfying Eqn. 3) integers x_1, x_2, y_1, y_2. *Task:* Decide whether or not $S \stackrel{?}{=} y_1 x_1 G + y_1 x_2 + y_2 \bmod p$.

Security requires this problem to be hard, meaning that all polynomial-time quantum adversaries decide the RDH problem with a success probability negligibly far from a random guess. In particular, this implies that SSSLE is hard as well, because it is easy to solve RDH with an SSSLE oracle. In fact, it is not clear how to solve RDH without solving SSSLE and indeed Section 4.3 only considers attacks on the latter problem.

4.2. IND-CCA2

I claim that Ramstake offers IND-CCA2 security but make this claim without proof. The intuition is that if the RDH holds, then the encapsulator's view of the SNOTP is a perfect one-time pad. The transmitted seed is therefore hidden perfectly. Moreover, if both the RDH and SSSLE are hard, then the attacker has no way of knowing the encapsulator's short-and-sparse integers a' and b'. While the adversary does know the value of the SNOTP as padded with the multiple codeword, he needs to know b' in order to obtain the symmetric key. An adversary who successfully forges a new ciphertext with a' and b' meaningfully linked to the same inputs of a previous ciphertext, then the decapsulation algorithm recovers these values and can use them to solve RDH.

4.3. Attacks

This section covers a number of attacks on SSSLE. The lattice and algebraic attacks do not seem to apply or are otherwise wildly inefficient. Brute force seems to be the best performing attack. Consequently, parameters are set to make brute force sufficiently complex. Nevertheless, there may be better attacks than brute force and if they are discovered a readjustment of the parameters is recommended.

4.3.1. Lattice Reduction

Lattice basis reduction algorithms such as LLL excel at finding short solutions to linear equations. However, in the case of SSSLE, the shortness is not the only distinguishing feature of the solutions; they must be sparse as well. In fact, LLL will find a solution whose length (in the ℓ^2 norm) is on the order of $p^{1/2}$, whereas the solution to the SSSLE problem has length between the orders of $p^{3/4}$ and $p^{1/2}$. However, it is far from unique: the same linear system of equations has roughly $p^{1/4}$ other solutions inside this range. LLL does not favor sparse solutions over short ones.

4.3.2. Algebraic System Solving

It is possible in theory to formulate the shortness-and-sparsity constraint algebraically, by constructing polynomials that evaluate to zero in all points that satisfy

the constraint. At this point a Gröbner basis algorithm can be used to compute a solution. However, the degree of this constraint polynomial is infeasibly large, roughly $\binom{3\pi/4}{\varsigma}$. Constructing it requires more work than exhaustively enumerating all potential solutions and testing to see if the linear equations are satisfied.

Another option is to treat the coefficients of the balanced binary expansion of the solutions, as variables in and of themselves. This strategy requires adding polynomials to require that each coefficient lie in $\{-1, 0, 1\}$, and that at most ς of them are different from zero. The result is a nonlinear system of roughly $\frac{3}{2}\pi$ equations in as many variables. This task is infeasible as well.

4.3.3. Brute Force

A brute force strategy to attack SSSLE is rather straightforward, especially in the context of Ramstake where $n = 2$. Choose a random short-and-sparse assignment for x_1, compute x_2 from the given information and test if it is also short-and-sparse. Assuming the solution is unique, the success probability of a single trial is one over the search space:

$$\Pr[\text{success}] = \binom{\pi/4}{1}\binom{3\pi/4}{\varsigma}2^{\varsigma} \ . \tag{4}$$

In the case of Ramstake RS 22040, the logarithm base 2 of this number is -272.76. Grover's algorithm offers the usual square-root speedup, making for a quantum attack complexity of $2^{136.38}$.

5. Advantages and Limitations

Advantage: Simplicity. Simplicity is the key selling point of Ramstake. Simple schemes are easier to implement, easier to debug, and easier to analyze. While simpler schemes are sometimes also easier to break, the a scheme's resilience to attacks should not rely on its complexity.

Limitation: New Hard Problem. The hard problem on which Ramstake relies is new and understudied. As a result, it does not offer much assurance of security compared to schemes that have existed (and remained unbroken) for much longer.

Advantage: Problem Diversity. On the other hand, breakthroughs in cryptanalysis or hard problem solving that break or severely harm other schemes may leave Ramstake intact.

Advantage/Limitation: Bandwidth and Speed. This point must be assessed in comparison to other KEM proposals. While lattice-based KEMs are likely to be faster and to require less bandwidth, Ramstake might still be competitive to or even outperform other schemes in this regard.

A. IP Statement

A.1. Statement by Submitter

I, Alan Szepieniec, of Kasteelpark Arenberg 10 / 3001 Heverlee / Belgium , do hereby declare that the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Ramstake, is my own original work, ~~or if submitted jointly with others, is the original work of the joint submitters~~.

I further declare that (check one):

- ✓ I do not hold and do not intend to hold any patent or patent application with a claim which may cover the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Ramstake; OR (check one or both of the following):

- ☐ to the best of my knowledge, the practice of the cryptosystem, reference implementation, or optimized implementations that I have submitted, known as Ramstake, may be covered by the following U.S. and/or foreign patents: "none";

- ☐ I do hereby declare that, to the best of my knowledge, the following pending U.S. and/or foreign patent applications may cover the practice of my submitted cryptosystem, reference implementation or optimized implementations: "none".

I do hereby acknowledge and agree that my submitted cryptosystem will be provided to the public for review and will be evaluated by NIST, and that it might not be selected for standardization by NIST. I further acknowledge that I will not receive financial or other compensation from the U.S. Government for my submission. I certify that, to the best of my knowledge, I have fully disclosed all patents and patent applications which may cover my cryptosystem, reference implementation or optimized implementations. I also acknowledge and agree that the U.S. Government may, during the public review and the evaluation process, and, if my submitted cryptosystem is selected for standardization, during the lifetime of the standard, modify my submitted cryptosystem's specifications (e.g., to protect against a newly discovered vulnerability).

I acknowledge that NIST will announce any selected cryptosystem(s) and proceed to publish the draft standards for public comment.

I do hereby agree to provide the statements required by Sections 2.D.2 and 2.D.3 in the Call For Proposals for any patent or patent application identified to cover the practice of my cryptosystem, reference implementation or optimized implementations and the right to use such implementations for the purposes of the public review and evaluation process.

I acknowledge that, during the post-quantum algorithm evaluation process, NIST may remove my cryptosystem from consideration for standardization. If my cryptosystem (or the derived cryptosystem) is removed from consideration for standardization or withdrawn from consideration by all submitter(s) and owner(s), I understand that rights granted and assurances made under Sections 2.D.1, 2.D.2

and 2.D.3 of the Call For Proposals, including use rights of the reference and optimized implementations, may be withdrawn by the submitter(s) and owner(s), as appropriate.

Signed: Alan Szepieniec
Title: ir.
Date:
Place:

A.2. Statement By Implementation Owner

I, Alan Szepieniec, Kasteelpark Arenberg 10 / 3001 Heverlee / Belgium, am the owner ~~or authorized representative of the owner (print full name, if different than the signer)~~ of the submitted reference implementation and optimized implementations and hereby grant the U.S. Government and any interested party the right to reproduce, prepare derivative works based upon, distribute copies of, and display such implementations for the purposes of the post-quantum algorithm public review and evaluation process, and implementation if the corresponding cryptosystem is selected for standardization and as a standard, notwithstanding that the implementations may be copyrighted or copyrightable.

Signed: Alan Szepieniec
Title: ir.
Date:
Place:

References

[1] Aggarwal, D., Joux, A., Prakash, A., Santha, M.: A new public-key cryptosystem via mersenne numbers. IACR Cryptology ePrint Archive 2017, 481 (2017), http://eprint.iacr.org/2017/481

[2] Aguilar, C., Gaborit, P., Lacharme, P., Schrek, J., Zémor, G.: Noisy diffie-hellman protocols (2010), https://pqc2010.cased.de/rr/03.pdf, PQCrypto 2010 The Third International Workshop on Post-Quantum Cryptography (recent results session)

[3] Aguilar, C., Gaborit, P., Lacharme, P., Schrek, J., Zémor, G.: Noisy diffie-hellman protocols or code-based key exchanged and encryption without masking (2010), https://rump2010.cr.yp.to/fae8cd8265978675893352329786cea2.pdf, CRYPTO 2010 (rump session)

[4] Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Newhope without reconciliation. IACR Cryptology ePrint Archive 2016, 1157 (2016), http://eprint.iacr.org/2016/1157

[5] Cramer, R., Shoup, V.: Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. IACR Cryptology ePrint Archive 2001, 108 (2001), http://eprint.iacr.org/2001/108

[6] Szepieniec, A., Preneel, B.: Short solutions to nonlinear systems of equations (2017), https://asz.ink/wp-content/uploads/2017/09/ssne.pdf, Number Theory Methods in Cryptography Conference, Warsaw