Efficient Post-quantum Digital Signature

DEEPSEC

Maksim Iavich

Private Key Encryption

ATTACKER

m messag e

C cyphe r

K key

K key

ALICE

BOB

$c := Enc_k(m)$

Encryptio n

$Dec_k(Enc_k(m)) = m$

$m := Dec_k(c)$

Decryptio n

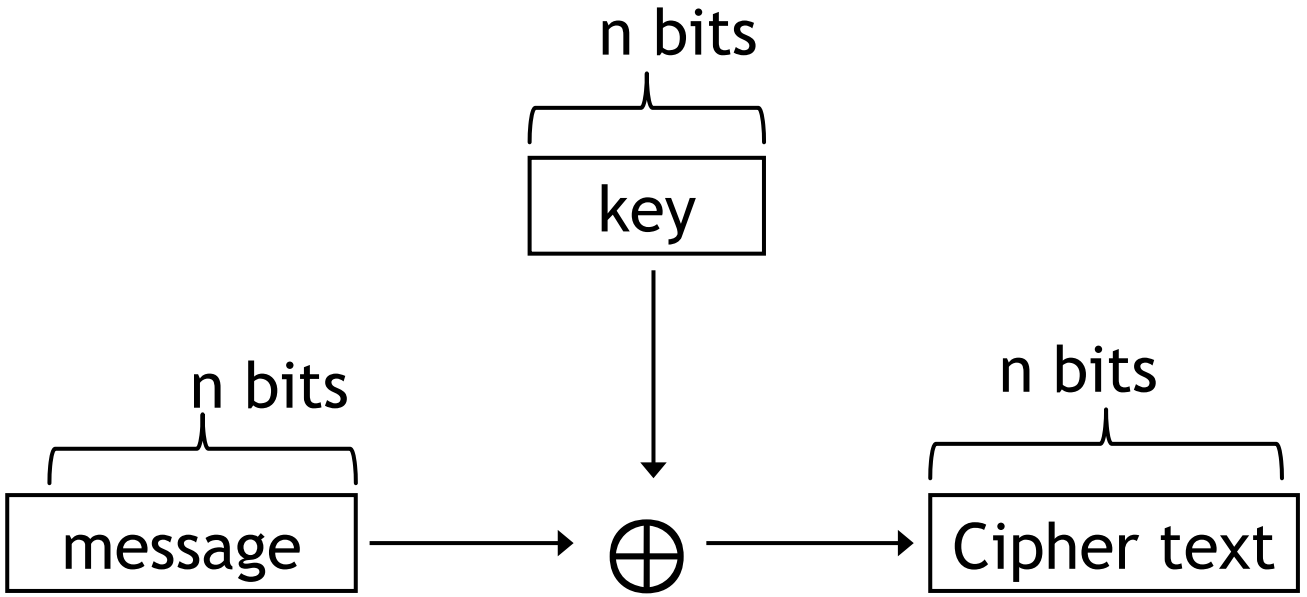
One-time pad

- $M = \{0,1\}^n$
- Gen: $k \in \{0,1\}^n$
- $Enc_k(m) = k \oplus m$
- $Dec_k(c) = k \oplus c$

- Truth:
  $Dec_k(Enc_k(m)) = k \oplus (k \oplus m)$
  $= (k \oplus k) \oplus m = m$

$$0 \oplus 1 = 1$$
$$0 \oplus 0 = 0$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 1$$
$$A \oplus A = 0$$

One-time pad



One-time pad

Problems:

- The size of the key must be the same as message (quite large)
- Is secure if one key is used once to decrypt only one message.

Use key twice ?

- $c_1 = k \oplus m_1$
  $c_2 = k \oplus m_2$

- Attacker is able:
  $c_1 \oplus c_2 = (k \oplus m_1) \oplus (k \oplus m_2) = m_1 \oplus m_2$

- Leaks information about m_1 and m_2

Using the same key twice?

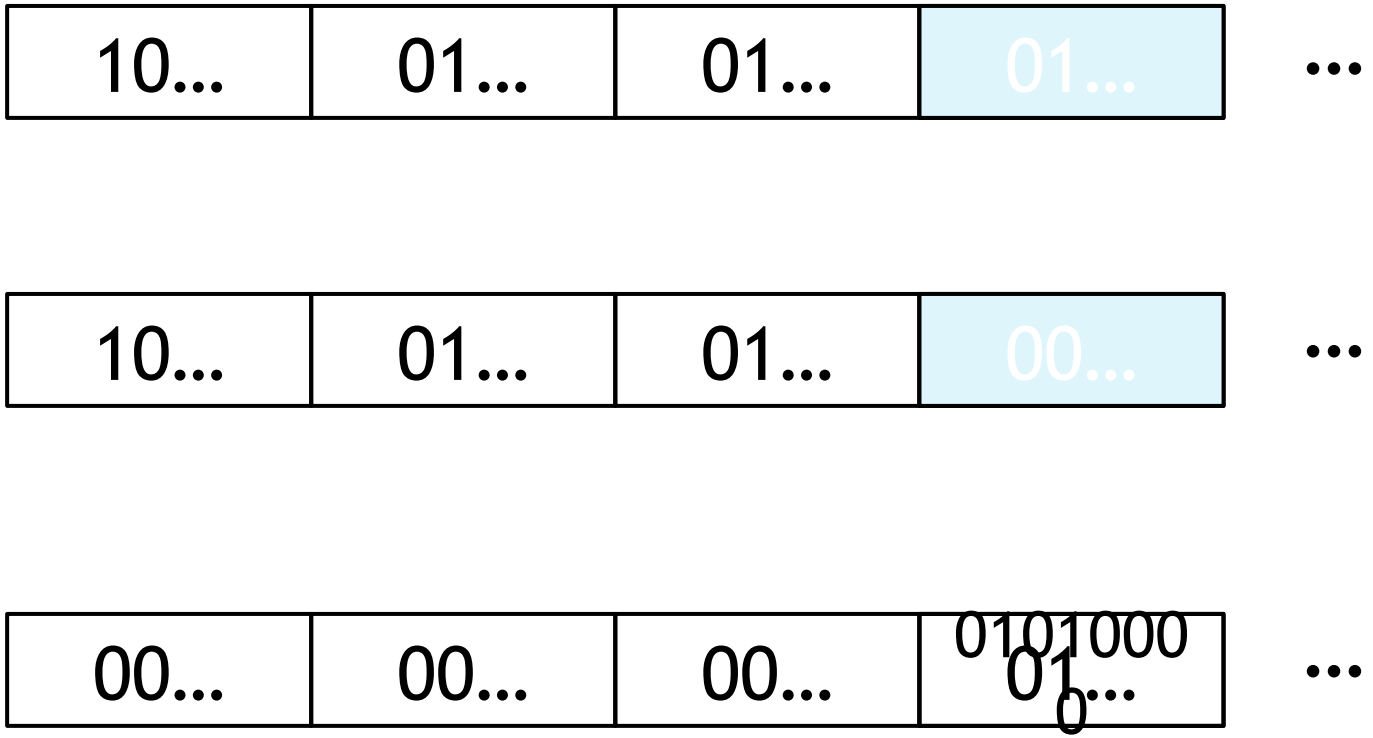
- $m_1 \oplus m_2$ is information about m_1, m_2
- Is this significant?
  - No longer perfectly secret!
  - $m_1 \oplus m_2$ reveals where m_1, m_2 differ
  - Frequency analysis
  - Exploiting characteristics of ASCII…

Hex	Dec	Char		Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char
0x00	0	NULL	null	0x20	32	Space	0x40	64	@	0x60	96	`
0x01	1	SOH	Start of heading	0x21	33	!	0x41	65	A	0x61	97	a
0x02	2	STX	Start of text	0x22	34	"	0x42	66	B	0x62	98	b
0x03	3	ETX	End of text	0x23	35	#	0x43	67	C	0x63	99	c
0x04	4	EOT	End of transmission	0x24	36	$	0x44	68	D	0x64	100	d
0x05	5	ENQ	Enquiry	0x25	37	%	0x45	69	E	0x65	101	e
0x06	6	ACK	Acknowledge	0x26	38	&	0x46	70	F	0x66	102	f
0x07	7	BELL	Bell	0x27	39	'	0x47	71	G	0x67	103	g
0x08	8	BS	Backspace	0x28	40	(	0x48	72	H	0x68	104	h
0x09	9	TAB	Horizontal tab	0x29	41	)	0x49	73	I	0x69	105	i
0x0A	10	LF	New line	0x2A	42	*	0x4A	74	J	0x6A	106	j
0x0B	11	VT	Vertical tab	0x2B	43	+	0x4B	75	K	0x6B	107	k
0x0C	12	FF	Form Feed	0x2C	44	,	0x4C	76	L	0x6C	108	l
0x0D	13	CR	Carriage return	0x2D	45	-	0x4D	77	M	0x6D	109	m
0x0E	14	SO	Shift out	0x2E	46	.	0x4E	78	N	0x6E	110	n
0x0F	15	SI	Shift in	0x2F	47	/	0x4F	79	O	0x6F	111	o
0x10	16	DLE	Data link escape	0x30	48	0	0x50	80	P	0x70	112	p
0x11	17	DC1	Device control 1	0x31	49	1	0x51	81	Q	0x71	113	q
0x12	18	DC2	Device control 2	0x32	50	2	0x52	82	R	0x72	114	r
0x13	19	DC3	Device control 3	0x33	51	3	0x53	83	S	0x73	115	s
0x14	20	DC4	Device control 4	0x34	52	4	0x54	84	T	0x74	116	t
0x15	21	NAK	Negative ack	0x35	53	5	0x55	85	U	0x75	117	u
0x16	22	SYN	Synchronous idle	0x36	54	6	0x56	86	V	0x76	118	v
0x17	23	ETB	End transmission block	0x37	55	7	0x57	87	W	0x77	119	w
0x18	24	CAN	Cancel	0x38	56	8	0x58	88	X	0x78	120	x
0x19	25	EM	End of medium	0x39	57	9	0x59	89	Y	0x79	121	y
0x1A	26	SUB	Substitute	0x3A	58	:	0x5A	90	Z	0x7A	122	z
0x1B	27	FSC	Escape	0x3B	59	;	0x5B	91	[	0x7B	123	{
0x1C	28	FS	File separator	0x3C	60	<	0x5C	92	\	0x7C	124	|
0x1D	29	GS	Group separator	0x3D	61	=	0x5D	93	]	0x7D	125	}
0x1E	30	RS	Record separator	0x3E	62	>	0x5E	94	^	0x7E	126	~
0x1F	31	US	Unit separator	0x3F	63	?	0x5F	95	_	0x7F	127	DEL

Source: http://benborowiec.com/2011/07/23/better-ascii-table/

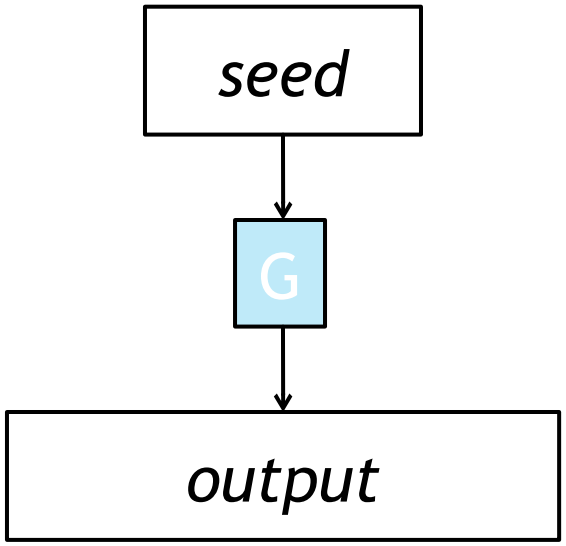
- Letters all begin with 01…
- The space character begins with 00…
- XOR of two letters gives 00…
- XOR of letter and space gives 01…
- Easy to identify XOR of letter and space!

In pictures…

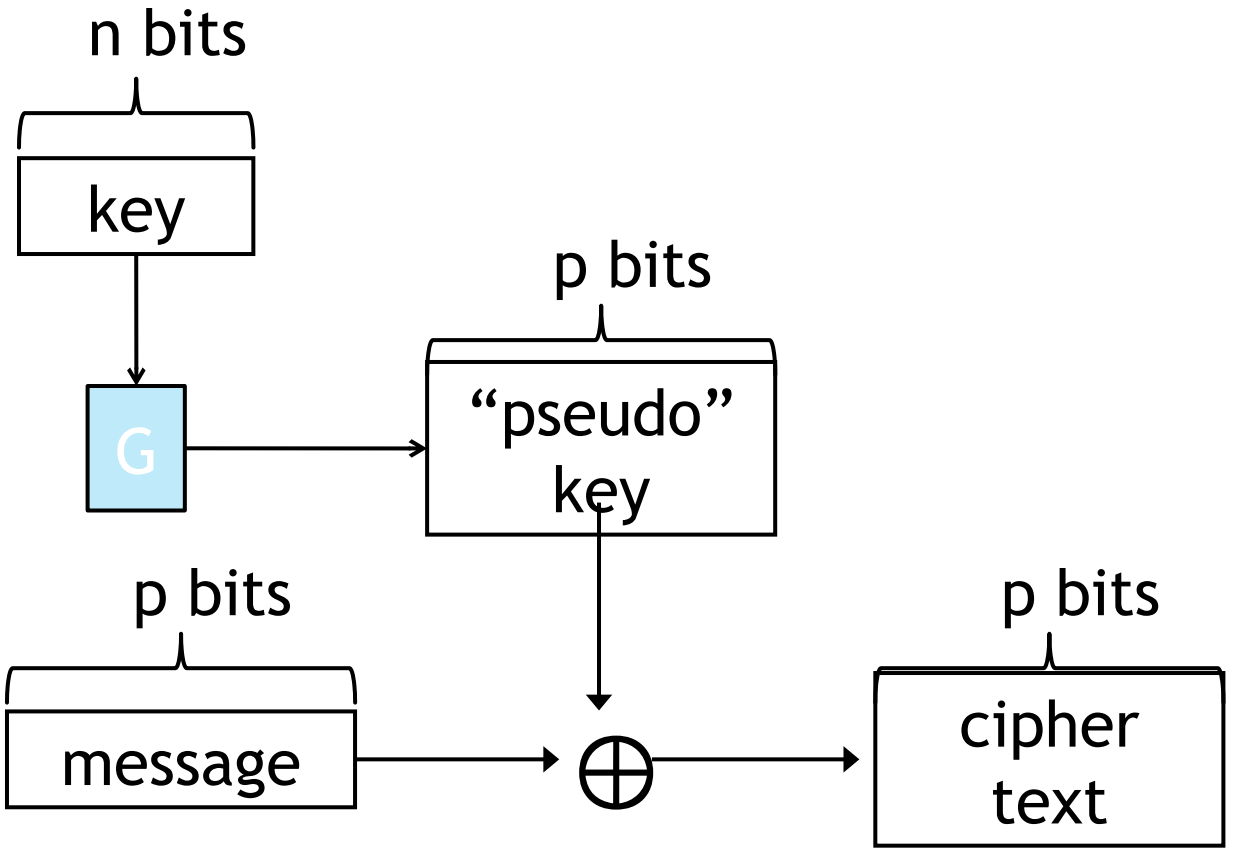


PRGs

- G – defined polynomial time algorithm
- G increases: $|G(x)| = p(|x|) > |x|$

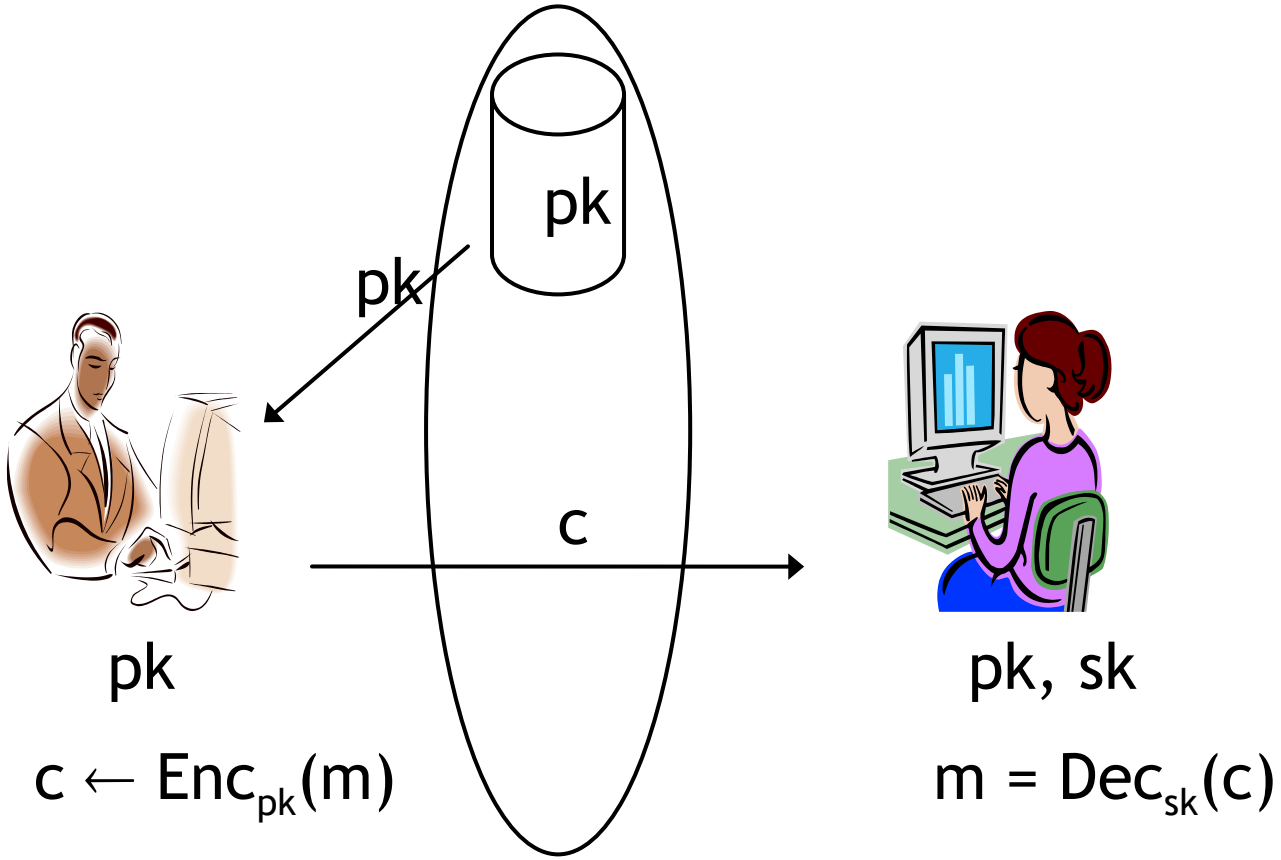


“Pseudo” one-time pad



By means of this key size is increased

Public-key encryption



“Plain” RSA encryption

N, e

c

$(N, e, d) \leftarrow \text{RSAGen}(1^n)$

$pk = (N, e)$

$sk = d$

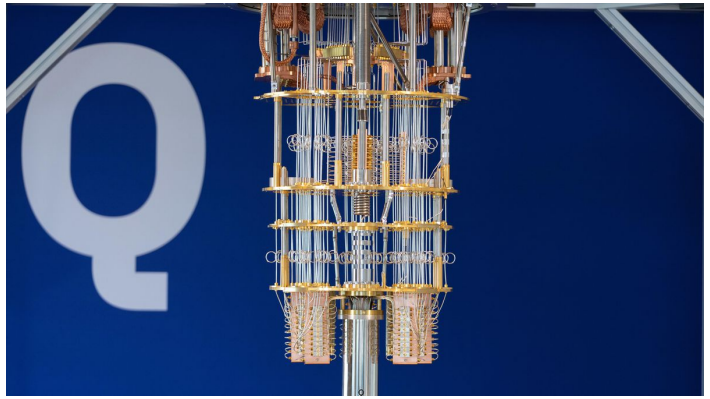
$c = [m^e \bmod N]$

$m = [c^d \bmod N]$

Digital signatures

- Digital signatures have become a key technology for making the Internet and other IT infrastructures secure. Digital signatures provide authenticity, integrity and non-repudiation of data. Digital signatures are very widely used in the identification and the authentication protocols. So, the existence of secure digital signature algorithms is obligatory for cyber security. The digital signature algorithms that are used in practice today are RSA , DSA and ECDSA .

Quantum computers



- GOOGLE Corporation, in conjunction with with the company D-Wave signed contract about creating quantum computers. D-Wave 2X - is the newest quantum processor, which contains physical qubits.
- Each additional qubit doubles the data search area, thus is also significantly increased the calculation speed. Quantum computers will destroy systems based on the problem of factoring integers (e.g., RSA). RSA cryptosystem is used in different products on different platforms and in different areas.

RSA system is widely used in operating systems from Microsoft, Apple, Sun, and Novell. In hardware performance RSA algorithm is used in secure phones, Ethernet, network cards, smart cards, and is also widely used in the cryptographic hardware. Along with this, the algorithm is a part of the underlying protocols protected Internet communications, including S / MIME, SSL and S / WAN, and is also used in many organizations, for example, government, banks, most corporations, public laboratories and universities.

News from Google

- Google made a huge revelation on October 23, 2019, when it announced that it had reached something called “quantum supremacy.” Via an article in the journal Nature, Google said their quantum computer, called Sycamore, solved a particularly difficult problem in 200 seconds. For comparison, Google said the world’s current fastest classical computer — one called Summit owned by IBM that’s as big as two basketball courts — would take 10,000 years to solve that same problem. This is what “quantum supremacy” means. It’s when a quantum computer — one that runs on the laws of quantum physics as opposed to the classical computers we’re familiar with (i.e. phones and laptops), which run on classical physics like Newton’s laws of motion — does something that no conventional computer could do in a reasonable amount of time.

IBM’s answer

- IBM responded to Google’s news to say that actually, Summit could solve the quantum computers’ problem in two and a half days — not 10,000 years as Google had suggested. In this episode of Recode’s Reset podcast, host Arielle Duhaime-Ross and Kevin Hartnett, a senior writer for the math and physics magazine Quanta, break down exactly what quantum computing is and why Google dunking on IBM both was and wasn’t a huge deal.

RSA ALTERNATIVES

Hash-based Digital Signature Schemes:

A code-based public-key encryption system

Lattice-based Cryptography: proofs are based on worst-case hardness.

Multivariate public key cryptosystem – MPKCs:

Successful attacks

- To date are already found successful attacks on this crypto system.
- The Ph.D. candidate of Dublin City University (DCU) Neill Costigan with the support of Irish Research Council for Science, Engineering and Technology (IRCSET), together with professor Michael Scott, Science Foundation Ireland (SFI) member successfully were able to carry out an attack on the McEliece algorithm. To do this they needed 8,000 hours of CPU time. In the attack representatives of four other countries took part. Scientists have discovered that the initial length of the key in this algorithm is insufficient and should be increased.
- This system cannot be also used to encrypt the same message twice and to encrypt the message when is known it’s relation with the other message.

- Should be noted the importance of efficiency spectrum. To date experts have reached quite good results in the speed algorithm processing. According to the investigation results it becomes clear that the proposed post-quantum cryptosystems are relatively little effective. Implementation of the algorithms requires much more time for their processing and verification.
- Inefficient cryptography may be acceptable for the general user, but it cannot be acceptable for the internet servers that handle thousands of customers in the second. Today, Google has already has problems with the current cryptography. It is easy to imagine what will happen when implementing crypto algorithms will take more time.
- The development and improvement of modern cryptosystems will take years. Moreover, all the time are recorded successful attacks on them. When is determined the encryption function, and it becomes standard, it needs the appropriate implementation of the corresponding software, and in most cases, hardware.

RSA ALTERNATIVES – HASH BASED

- Traditional digital signature systems that are used in practice are vulnerable to quantum computers attacks. The security of these systems is based on the problem of factoring large numbers and calculating discrete logarithms. Scientists are working on the development of alternatives to RSA, which are protected from attacks by quantum computer. One of the alternatives are hash based digital signature schemes. These systems use a cryptographic hash function. The security of these digital signature systems is based on the collision resistance of the hash functions that they use.

LAMPORT-DIFFIE ONE-TIME SIGNATURE SCHEME (KEY GENERATION)

- Keys generation in this system occurs as follows: the signature key X of this system consists of 2n lines of length n, and is selected randomly.
- $X = (x_{n-1}[0], x_{n-1}[1], \ldots, x_0[0], x_0[1]) \in \{0,1\}^{n,2n}$
- Verification key Y of this system consists of 2n lines of length n.
- $Y = (y_{n-1}[0], y_{n-1}[1], \ldots, y_0[0], y_0[1]) \in \{0,1\}^{n,2n}$
- This key is calculated as follows:
- $y_i[j] = f(x_i[j]), 0<=i<=n-1, j=0,1$
- f – is one-way function:
- $f: \{0,1\}^n \rightarrow \{0,1\}^n$;

DOCUMENT SIGNATURE

- To sign a message m of arbitrary size, we transform it into size n using the hash function:
- $h(m)=hash = (hash_{n-1}, \dots , hash_0)$
- Function h- is a cryptographic hash function:
- $h: \{0,1\}^* \rightarrow \{0,1\}^n$
- The signature is done as follows:
- $sig= (x_{n-1}[hash_{n-1}], \dots, x_0[hash_0]) \in \{0,1\}^{n,n}$
- i-th string in this signature is equals to $x_i[0]$, if i-th bit in sign is equal to 0. The string is equal to $x_i[1]$, if i-th bit in sign is equal to 1.
- Signature length is $\mathbf{n^2}$.

DOCUMENT VERIFICATION

To verify the signature sig = (sig_{n-1}, …, sig_0), is calculated hash of the message hash = ($hash_{n-1}$, … , $hash_0$) and the following equality is checked:

($f(sig_{n-1})$, …, $f(sig_0)$) = ($y_{n-1}[hash_{n-1}]$, …, $y_0[hash_0]$)

If the equation is true, then the signature is correct.

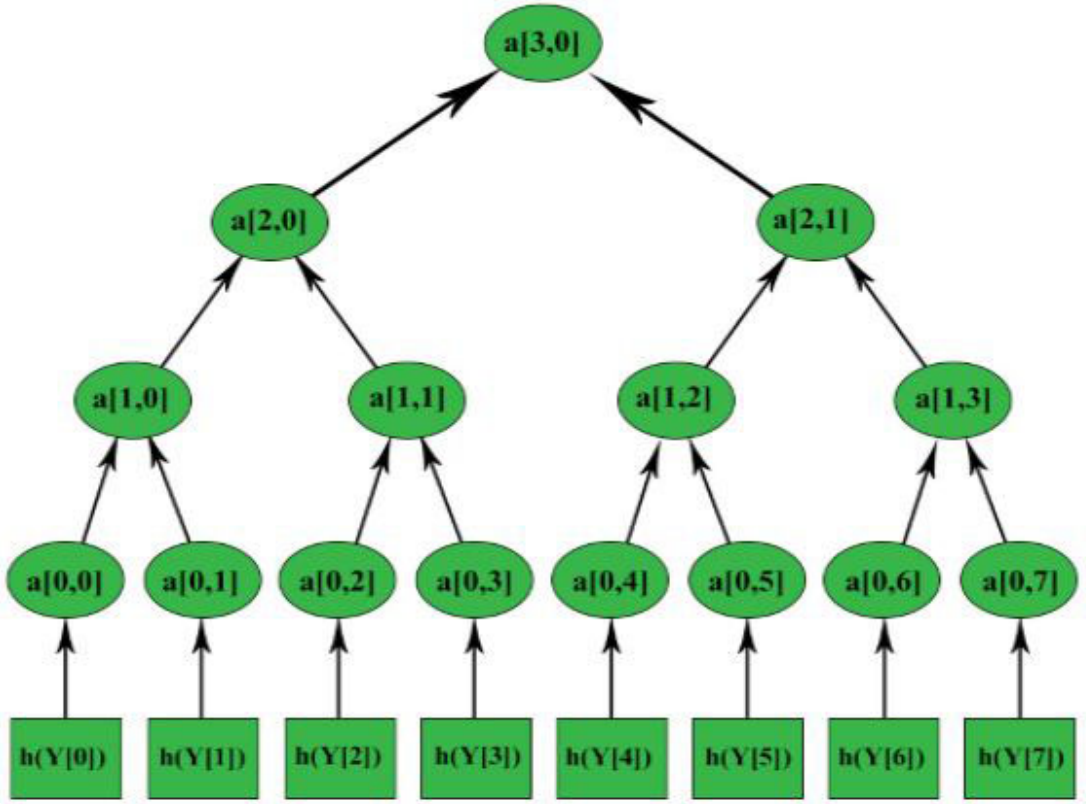
WINTERNITZ ONE TIME SIGNATURE SCHEME. KEY GENERATION

To achieve security $O(2^{80})$, the total size of public and private keys must be $160*2*160$ bits = 51200 bits, that is 51200/1024=50 times larger than in the case of RSA. We must also note that the size of the signature in the given scheme is much larger than in the case of RSA. Winternitz One-time Signature Scheme was proposed to reduce the size of the signature.

MERKLE

- One-time signature schemes are very inconvenient to use, because to sign each message, you need to use a different key pair. Merkle crypto-system was proposed to solve this problem. This system uses a binary tree to replace a large number of verification keys with one public key, the root of a binary tree. This cryptosystem uses an one-time Lamport or Winternitz signature scheme and a cryptographic hash function:
- $h:\{0,1\}^* \rightarrow \{0,1\}^n$
- **Key generation:** The length of the tree is chosen $H>=2$, with one public key it is possible to sign 2^H documents. 2^H signature and verification key pairs are generated; X_i, Y_i, $0<=i<=2^H$. X_i- is signature key, Y_i- is verification key. $h(Y_i)$ are calculated and are used as the leaves of the tree. Each tree node is a hash value of concatenation of its children.

MERKLE TREE


a[3,0]
a[2,0]
a[2,1]
a[1,0]
a[1,1]
a[1,2]
a[1,3]
a[0,0]
a[0,1]
a[0,2]
a[0,3]
a[0,4]
a[0,5]
a[0,6]
a[0,7]
h(Y[0])
h(Y[1])
h(Y[2])
h(Y[3])
h(Y[4])
h(Y[5])
h(Y[6])
h(Y[7])

SIGNATURE GENERATION

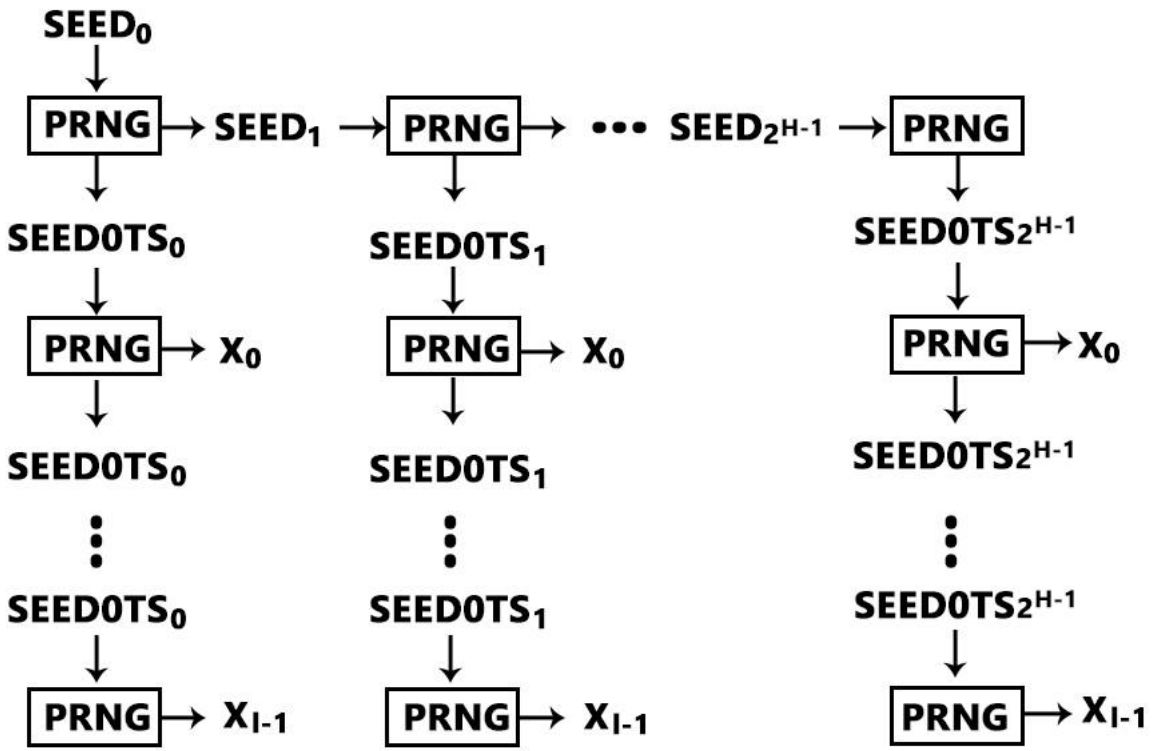
- To sign a message m of arbitrary size we transform it into size n using the hash function
- h (m) = hash, and generate an one-time signature using any one-time key X_{any}, the document's signature will be the concatenation of: one time signature, one-time verification key Y_{any}, index any and all fraternal nodes $auth_i$ in relation to Y_{any}.
- Signature= (sig||pub||any|| Y_{any}||$auth_0$,…,$auth_{H-1}$)
- **Signature verification:**
- To verify the signature we check the one-time signature of sig using Y_{any}, if it is true, we calculate all the nodes a [i, j] using "$auth_i$", index "any" and Y_{any}. We compare the last node, the root of the tree with public key, if they are equal, then the signature is correct.

	Lamport	Winternitz	Merkle
Use f to generate keys	2n	$p(2^w-1)$	$2^{H+1}-1$
Use f to calculate the signature	Is not used	$p(2^w-1)$	
Use f to generate verify the signature	n	$p(2^w-1)$	

PRNG INTEGRATION

- To generate a public key you need to calculate and store 2^H pairs of one-time keys. Storing this amount of information is not effective in practice. In order to save space, it was suggested to use the PRNG random number generator. When using PRNG, it is sufficient to store only the seed of the generator and use it to generate one-time keys. It is necessary to calculate one-time keys twice: once in the key generation stage and then in the signature stage of the message. PRNG receives a seed of length n and outputs a new seed and a random number of length n.
- PRNG : $\{0, 1\}^n$ ▭ : $\{0, 1\}^n$ x $\{0, 1\}^n$
- **Key generation using PRNG:**
- We choose randomly the seed s_0 of length n, using s_i we work out sot_i, as following:
- $PRNG(s_i) = (sot_i , s_{i+1})$ $0 \leq i < 2^H$
- sot_i changes each time when PRNG launches. For X_i key calculation it is enough to know only s_i.

PRNG INTEGRATION



CSPRNGs

- Pseudorandom number generators are widely used in Cryptography. This type of PRNGs are called cryptographically secure pseudorandom number generators CSPRNGs. Blum and Micali (Blum and Micali, 1984) and the Blum Blum Shub generators (Blum et al., 1986) are often used in cryptography applications. These CSPRNGs are based on number theory. Blum Blum Shub works as follows: the output bits come from the recursive formula $X_{i+1} = X^2_i \mod N$ for $N = pq$ the product of two primes p and q congruent to 3 mod 4.
- X_i is the ith number used as the internal state. The algorithm has N and X_0 as inputs and the ith output bit is the parity of X_i. The initial state X_0 should come from a TRNG.

Breaking PRNG

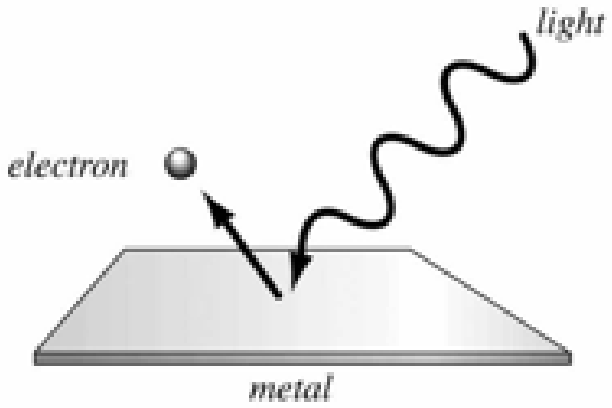
- Quantum computers are able to crack PRNG, which were considered safe against attacks of classical computers. A polynomial quantum time attack on PRNG Blum-Micali is shown. This PRNG is considered safe from threats of standard computers. This attack uses Grover algorithm along with the quantum discrete logarithm, and is able to restore the values at the generator output for this attack. The attacks like these represent a threat of cracking PRNG, used in many real-world crypto systems. As we see, Merkle crypto system with built-in PRNG can be vulnerable to attacks of quantum computers.

HASH_DBRG and HMAC_DBRG

- As a CSPRNG in Merkle we offer an algorithm based on a hash function, as the whole the algorithm is based on it. NIST has recommended two continuous hash based PRNGs: HASH_DBRG and HMAC_DBRG. We offer to use HASH_DBRG as it is more efficient.
- We offer to use physical quantum random number generator (QRNG) for generating the seed for HASH_DBRG.

QRNGs

- In 1961 the researchers offered to use quantum phenomena as a source of randomness . Afterwards the researchers began to work actively on it. Radioactive decay was a particularly accessible source of true randomness. Geiger-Muller tubes were already sensitive enough to capture and amplify α, ß and γ radiation, well-characterized radioactive samples were available. Almost all radioactivity-based QRNGs were based on the detection of ß radiation.



Radioactivity-based quantum random number generators

- In a Geiger-Muller detector, a single particle makes an ionization event that is amplified in a Townsend avalanche. We get is a device that creates a pulse for each detected particle. Any concrete atom's probability to decay in a time interval (t, t + dt) can be presented as exponential random variable, $Pr(t)dt = \lambda_n e^{(-\lambda_n t)dt}$, where λ_n is a decay constant. The time between detected pulses is an exponential random variable, if the sample saves a lot of original atoms and the detector system does not change in this time interval. The times values are independent from previous ones. The number of pulses that occur in a concrete time period follows a Poisson distribution. It can be shown experimentally, that the pulses occur at independent times. The probability of finding n pulses in an period of T seconds is $P_n(T) = (\lambda T)^n / n! * e-\lambda T$, where λ is a mean number of pulses detected in one second and corresponds to the parameter of the exponential distribution.

- QRNGs based on radioactive decay have many common features. Mostly they use digital counters to convert the pulses from the detector into random values. The counter increases the values of its output by 1 when it gets a pulse as an input and the counter value can be reset to 0. Another important element is timing with a digital clock. If f is a frequency of the clock. A fast clock, with $f > \lambda$, generates a big amount of pulses and a slow clock, with $f < \lambda$, produces a pulse rarely, so many counts can be registered in detector. The randomness in the time of arrival can be converted into random values in some different ways. For example the generators of Isida and Ikeda and Vincent (Vincent, 1970)use a fast clock counter that is read and afterwards reset to zero every time we get a count on the detector. The value of the counter at detection time is used to output the random number. The distribution of values is not uniform so it must be corrected.

- Another option is to use a slow clock to determine when to read the counter. For example in the generator of Schmidt, the pulses from the detector increase the value of a counter. When the clock produces a new pulse, the value of the counter is used as a random digit and the count resets again to 0. The output corresponds to the number of particle counts in each clock period. In order to generate values from 0 to V−1, we use modulo V operation. If V= 2 we get a binary random number generator. The distribution of the values is not uniform, but if we take the modulo V addition of multiple outputs, we receive a distribution with very small bias. This procedure is called contraction.

Optical quantum random number generators

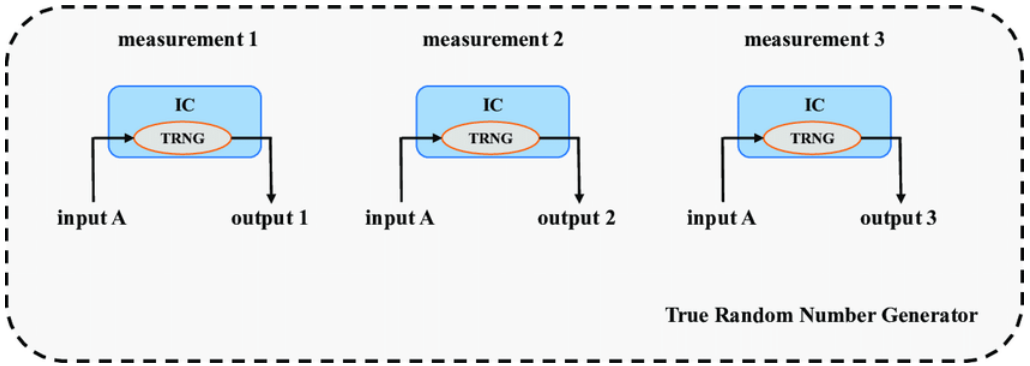
- Today, almost all the existing QRNGs are based on quantum optics. Most parameters of the quantum states of the light have rather good randomness property. It gives us the opportunity of good choice of implementations. Light from lasers, light emitting diodes or single photon sources is a convenient and rather affordable alternative of radioactive material as a source of quantum randomness and there exist many available detectors.
- Time of arrival generators, are representatives of optical quantum random number generators (OQRNGs). These OQRNGs are based on the same basic principles as the QRNGs that detect radioactive decay.

Time of arrival generators

- QRNG that uses time has a rather weak source of photons, it has a detector and timing circuitry that traces the time of each detection or the clicks number in a concrete time period. The detector receives photons from LED incoherent light and from the coherent states from a laser in an exponentially distributed time $\lambda e^{-\lambda t}$, where λ is an average number of photons per second. The time between two detections is exponential as it is the difference of two exponential random variables. We can compare the differences of the time between the arrival of consecutive pulses, we will get two time differences t_0 and t_1 so we can compare them also. In order to get the random bit, if $t_1 > t_0$ we assign a 1 and if $t_0 > t_1$ we can assign a 0. We take the integers with the number of the counted clock periods c_0 and c_1 instead of the real times t_0 and t_1. We can get the case, where $t_0 = t_1$, with some negligible probability for an ideal continuous time measurement and of course this case must be taken into the consideration.

- In 2007 year was offered the first optical quantum random number generator that uses time detection. It takes the photons from an LED arriving at a PMT after it compares the arrivals times.
- We offer to use a time arrival generator as a seed of HASH_DBRG.

Improvements



- **PHOTON COUNTING QUANTUM RANDOM NUMBER GENERATORS**

Assign more then one bit during each measurement

- **ATTENUATED PULSE QUANTUM RANDOM NUMBER GENERATORS**

Are based on a simplified version of the previous methods have fewer requirements for detectors.

- **Novel Generator**

We offer the improved quantum random number generator, which is based the on time of arrival QRNG. In time of arrival QRNGs at best, we get one random bit from each detected photon, this probability is reduced by the inefficiency of the detector or by dead time. In most cases, the frequency of random number generators is measured in Mbps, which is not enough for fast applications such as QKD. If we use multiple detectors to generate more random bits, we will have bias, generated by the different efficiencies of the detectors. Using one detector and comparing three successful detection time events, we rule out this bias. It is also rather convenient to use the simple version of the detectors with few requirements for this we offer to use the technology used in Attenuated pulse Quantum Random Number Generators.

SHA2

- SHA-2 is a family of cryptographic hash functions created by United States National Security Agency - NSA in 2001. These functions are constructed using Merkle-Damgård structure and the one-way compression function created by means of Davies–Meyer structure from the special block cipher. This family includes the following hash functions: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256. These functions use the different amount of shifts and additive constants, but they have the identical structure, which differs in the number of rounds.

Integration of the SHA-2 Hash Function into the Merkle Scheme

- It must be emphasized that efficiency is a very important aspect in cryptography. Considering that the encryption scheme uses the hash function many times, we analyzed the efficiency of Merkle's signature using the integrated SHA-512 hash function, and we compared it to the implementation that is based on the usage of the SHA-256 hash function. The scheme is defined by the steps, which are enumerated in the following pseudo code.
- The performance tests were performed on a machine that is powered by the processor i7-8565U CPU. Thus, the implementation that considers the SHA-256 hashing model produced a key generation time of 0.2522974 seconds, a signature time of 0.0004094000000000042seconds, and a verification time of 0.0025690000000000435 seconds. Furthermore, the implementation that considers the SHA-512 hashing model produced a key generation time of 0.3956368 seconds, a signature time of 0.0032782999999999562 seconds, and a verification time of 0.010349899999999912 seconds.

The BLAKE2 hash function

- The BLAKE2 is a family of cryptographic hash functions designed in 2012 by Jean-Philippe Aumasson, Samuel Neves, Christian Winnerlein and Zooko Wilcox-O'Hearn. This cryptographic scheme is considered to be rather efficient, as it is faster than MD5, SHA-1, SHA-2, and SHA-3. It is also considered rather secure, as its security is based on the immunity to length extension, and the in differentiability from a random oracle [17]. The BLAKE2 family includes two main functions: BLAKE2b and BLAKE2s. Thus, BLAKE2s is well optimized for the platforms of 8-bit and 32-bit, and it outputs hash values between 1 and 32 bytes long. Furthermore, BLAKE2b is well optimized for 64-bit platforms, which include NEON-enabled ARMs. It outputs the hash values of the size between 1 and 64 bytes. In SSSE3 (Supplemental Streaming SIMD Extensions 3)-capable CPUs, the arithmetic cost of the involved processing is reduced by 12%.

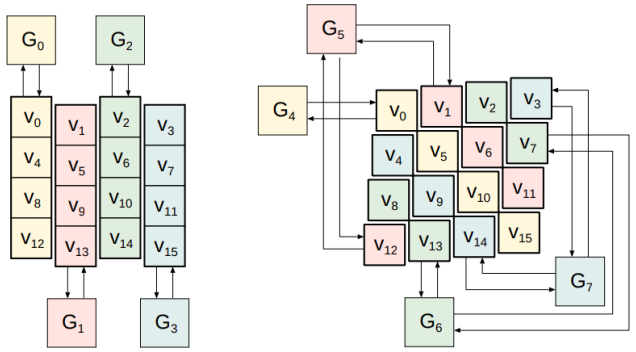


Figure 2.2: Column step and diagonal step.

The BLAKE3 hash function

- The BLAKE3 scheme is significantly more efficient than SHA-1, SHA-2, SHA-3, and BLAKE2. In the context of BLAKE3, the number of rounds is reduced from 10 to 7. Thus, most block ciphers specify a round, which includes the building blocks that are assembled together in order to design a cryptographic function, which runs multiple times. The BLAKE3 scheme considers a binary tree structure, in order to enable the use of parallelism, which is provided by SIMD instructions that are included in most of the modern central processing units. The BLAKE3 is a 128-bit security mechanism that can be used to prevent preimage, collision, or differentiability attacks.

Offered scheme

- **Key generation**
- The size of the tree must be H>=2 and using one public key 2^H document can be signed. Using novel generator we generate a seed. The PRNG HASH_DBRG take the seed as the input and are generated signature and verification keys; X_i, Y_i, 0<=i<=2H. X_i- is the signature key, Y_i- is the verification key. To get the leaves of the tree, signature keys are hashed using the hash function $h:\{0,1\}^* \rightarrow \{0,1\}^n$.
- To get the parent node, the concatenation of two previous nodes is hashed. The root of the tree is the public key of the signature - public.
- **Message signature**
- To sign a message of any size, it is transformed to size of n by means of hashing.
- h (m) = hash, to sign the message, is used an arbitrary one-time key X_{arb}. This key is calculated by means of PRNG HASH_DBRG using the same seed got from our novel generator. The signature is a concertation of: one-time signature, one-time verification key, index of a key and all brother nodes according to the selected arbitrary key with the index “arb”.
- Signature= $(sig||arb||Y_{arb}||auth_0,\ldots,auth_{H-1})$

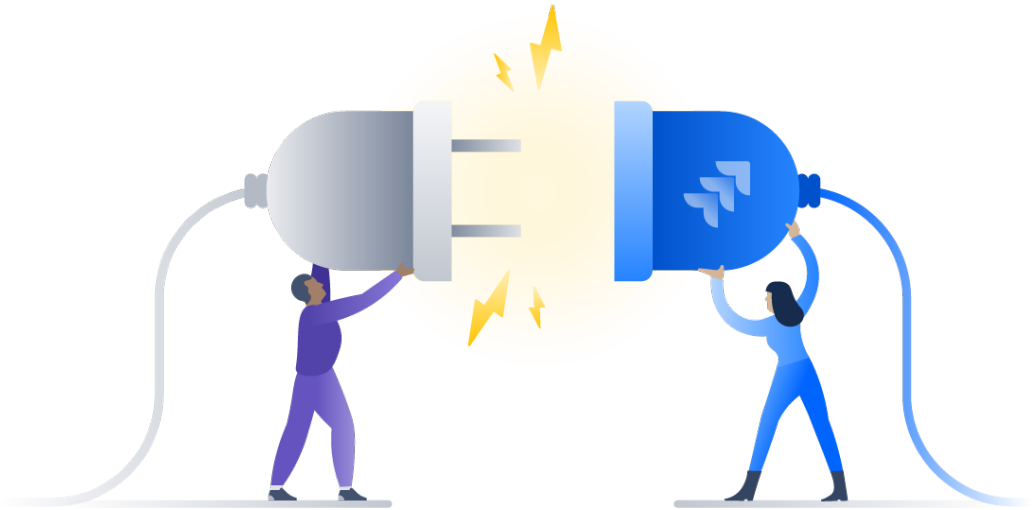
Signature verification

- To verify the signature, an one-time signature is checked using the selected verification key, if the verification has passed, all the needed nodes are calculated using "auth", index "arb" and Y_{arb}. If the root of the tree matches the public key, then the signature is correct.

Integrating BLAKE2b

- Blake2b is used as one-way function, as the hash function.

The scheme performance was assessed on the same machine with an i7-8565U CPU. Thus, the key generation time is 0.1879717 seconds, the signature time is 0.00029190000000006 seconds, and the verification time is 0.00128670000000008 seconds. The results show that in the case of the integration of BLAKE2b into the scheme, the system works even faster than in the case of SHA-256.



Security

- Nothing is changed in the classical version of Merkle. In order to improve the efficiency the hash based PRNG is integrated, the seed of PRNG is integrated by means of Quantum Random Number Generator. As we can see this PRNG is not vulnerable to attacks of quantum computers. As the hash function we use blake2b, which is secure against attacks of quantum computers.
- As we can see the proposed implementation of improved Merkle is secure.

THANK YOU! QUESTIONS?

MAKSIM IAVICH

SCIENTIFIC CYBER SECURITY ASSOCIATION ; CAUCASUS UNIVERSITY

T. +(995 595) 511355; E-mail: m.iavich@scsa.ge

www.scsa.ge

Scientific&practical cyber security journal – www.journal.scsa.ge

SCIENTIFIC CYBER SECURITY ASSOCIATION

კავკასიის უნივერსიტეტი
CAUCASUS UNIVERSITY