

Transfinite Cryptography

JACQUES PATARIN

Université de Versailles

45 avenue des Etats-Unis, 78035 Versailles Cedex, France[†]

Let assume that Alice, Bob, and Charlie, the three classical people of cryptography are not limited anymore to perform a finite number of computations on real computers, but are limited to α computations and to α bits of memory, where α is a fixed infinite cardinal. For example $\alpha = \aleph_0$ (the countable cardinal, i.e. the cardinal of \mathbb{N} the set of integers), or $\alpha = \mathfrak{C}$ (the cardinal of the set \mathbb{R} of real numbers). Is it possible to do secret-key cryptography? Public-key cryptography? Encryption? Authentication? Signatures? Is it possible to generalize the notion of one way function? The aim of this paper is to give some elements of answers to these questions. We will see for example that for secret-key cryptography there are some simple solutions. However for public-key cryptography the results are much less clear.

Key words: Cryptography with infinite computations, Generalizations of cryptographic problems and algorithms, Foundations and introduction to transfinite cryptography.

1 INTRODUCTION

Many parts of mathematics are used in modern cryptography: probability, logic, complexity theory, algebra, number theory, elliptic curves for example, and more. In fact, some cryptographers have even claimed that mathematics and cryptography may be the same subject (for example A. Albert [1]). However, from a practical point of view, modern mathematics are mainly built

[†] email: jacques.patarin@prism.uvsq.fr

from the analysis of the properties of infinite sets (real numbers, complex numbers, holomorphic functions, etc.). This is clearly different from cryptography where, from a practical point of view, 99.9 % of the mathematical results used come from the properties of finite sets (discrete mathematics). This is not very surprising since in cryptography the aim is to use real, finite, computers.

It is well known that some results about finite constructions can be obtained only by using infinite concepts (Goodstein theorem for example). Many results in number theory (about properties of integers) have been proved by using a natural framework with infinite notions (for example the prime number theorem, with holomorphic functions, even if it is possible to prove it only from the Peano arithmetic).

There are also some examples of use of infinite concepts to design finite cryptographic schemes, or to justify some cryptographic properties (see [12] or [13] for example, or the use of p-adic numbers).

The aim of this paper is different: in this paper, Alice, Bob, and Charlie, the three classical people of cryptography will be able to perform an infinite number of computations, but this number will be limited by an infinite fixed cardinal α . An analysis like this has been done before ([2], [16], [12] are the papers that I know) but in rather different models of infinite computations compared to the one we will use in this paper. In [16] the basic operations are limited, so in this model the function $x \mapsto x^2$ on the field \mathbb{Q} of rational numbers is one way. This model of computation of [16] seems to us a bit artificial, and in our (more natural) model of infinite computations, by performing \aleph_0 computations, we will easily be able to inverse $x \mapsto x^2$ on \mathbb{Q} .

In [2] and [12], it is also shown that there is no perfect encryption scheme on the set of an finite strings (on \mathbb{N} for example) without revealing some information on their length, unlike what we have on $[0, 1]$ for example. (Similarly for secret sharing problem). In [7] an example of encryption that can be performed on any finite string is shown (it can be used on \mathbb{N} for example).

In this paper we will have inputs on $[0, 1]$ (or larger sets) without looking for the case of the smaller infinite sets of inputs \mathbb{N} . This is because as soon as we have \aleph_0 bits of memory, it is possible to store any real value. Moreover we will analyze some problems not studied before in [2] and [12]: malleability, one way-functions, or in what ordinal order the messages will be sent for example. For secret-key cryptography, we will see that there are some simple solutions for stream ciphers, signatures, and authentication. As we will see, this paper is however still essentially an introduction to transfinite cryptography, since many natural problems are still open, particularly in public-key

transfinite cryptography.

2 OUR TRANSFINITE COMPUTATIONAL MODEL

Let α be a fixed infinite cardinal. For example $\alpha = \aleph_0$ (the countable cardinal, i.e. the cardinal of \mathbb{N}), or $\alpha = \mathfrak{C}$ (the continuum, i.e. cardinal of \mathbb{R}). Charlie, Bob, and Alice will be able to use “transfinite computers” able to perform α computations and able to use α bits of memory. It is possible to describe precisely this model of computations. Such a model has been specified in [14], [15], [3], [4], [5] for example. In order for this paper to be self contained, we will however quickly give here some details about such a possible model of computation. The general idea is to follow a generalization of the Church’s Thesis: as soon as a computation will be clearly feasible with α bits of memory and α computations, we will include it in the model. Moreover, the results of this paper will be very stable from little changes in the infinite computation model. Readers familiar with Ordinal Turing Machines, (OTM), with tapes whose cells are indexed by ordinals, as described in [4] can just not read this section 2 and go directly to section 3, since the new results of this paper are on transfinite cryptography, not on transfinite computers. We will speak of “ α programs”. We can assume that the memory is separated in 4 zones of bits: the input memory, the program memory, the variables of computation memory, and the output memory. Without loss of generality we can assume that the input memory is made of 1, or 2 (or more but $\leq \alpha$) inputs of α bits. The program memory contains a well ordered set of α elementary operations. Thanks to the fact that the program memory is well ordered, we can know at each “time” of the computation which is the next operation to perform. The word “time” if of course here a generalized word, it means that when any set of operations has been performed, we know precisely what is the next operation to be performed. More precisely than “time”, it is the succession of some ordinals that we will use. To each operation T at a certain place in the program we will associate an ordinal β , so we can say that T is the operation number β , or of position β . Each elementary operation can be of two kinds: simple, or GOTO. A simple elementary operation is a classic operation of present computer languages (such as C for example) on two words of 64 bits for example such that these two words are chosen at the addresses a and b of the memory and the result is stored at the address c of the memory. a and b can be addresses of the input memory, or of the variables of computation memory. c can be an address of the variables of computation memory, or of the output memory. a , b , and c are addresses of at most α

bits and are associated with the current operation. So each instruction of the program (operation and its position or number) can memorize a , b , c , and the operation to be performed. Of course, it is possible to use any other classical computer language instead of the C language, or to use words of 32 bits (or another length) instead of 64 bits. This will not change the set of functions that we can compute. A special instruction is the “stop instruction”. When this instruction is performed, the program stops and the output of the program is the value stored in the α bits of the output memory. The GOTO operation is an operation of the form (if $X = k$) then GOTO β where β is an ordinal. Thus this GOTO instruction says that the next instruction to be performed is the instruction number β (or of position β), if a variable X of α bits is equal to the value k of α bits. (Note that β can be the ordinal smaller than the ordinal of the current GOTO instruction performed). If $X \neq k$, to determine the next instruction we will follow, as for the simple instructions, the usual order of the ordinals of the instructions. It is also possible to describe our model of transfinite computations with generalized Turing machines.

Coding the instruction ordering

In the α bits of the program memory zone, there are various simple ways to describe the ordering (well ordering) of the instructions. Let us give here an example for $\alpha = \aleph_0$. (It is easy to generalize this example for any cardinal α). Let P be an α program. By definition, we will call “ordinal of P ” the ordinal of the (well ordered) set of all instructions of P . For example, if $\alpha = \aleph_0$, this ordinal may be ω , or ω^3 . A countable ordinal can be described as a good ordering on \mathbb{N} . So each countable ordinal can be written as a set of \aleph_0 integers: for each integer n , we will give the list of all the integers m such that $m < n$ for this ordering. We need for this $\leq \aleph_0 \times \aleph_0$ bits. The “infinite processor” can like this find the first instruction (no instruction is strictly smaller), and then, at each step, it can check all the integers in order to find the next instruction to be performed.

Remark. A classical result on ordinals is that countable ordinals has cardinal \aleph_1 (i.e. the smallest non countable infinite cardinal). We know that $\aleph_1 \leq \mathfrak{C}$. (However we do not know if $\aleph_1 = \mathfrak{C}$ or not, this is the famous undecidable problem called “continuum hypothesis”). Moreover each real number can be given by \aleph_0 bits. Therefore each countable ordinal can be given by \aleph_0 bits. This is what we do here for the ordinal of program P . What we have done here for $\alpha = \aleph_0$ can also be extended to any infinite cardinal α since for any infinite cardinal α , we have $\alpha^2 = \alpha$ (with the axiom of choice).

Example. The function $x \rightarrow x^2$ on \mathbb{Q} is a one way function in the model of infinite computation of [16]. In our model of infinite computations, this

function however is not a one way function. In order to find a rational (or a real) x such that $x^2 = y$ with \aleph_0 computations and \aleph_0 bits of memory, we can, for example, find all the bits of x , one by one. If we know that x is a rational number, then we can also try all the rational numbers one by one (card $\mathbb{Q} = \aleph_0$) square them, and see if we get y . Here again we need “only” \aleph_0 computations and \aleph_0 bits of memory.

Remark on the memory. On classical computers bits can have the value 0, or the value 1. In our model of computation, it is possible to assume that the values can be 0, 1, or “not fixed”. The value “not fixed” we will obtain for example when the bit has flipped from 0 to 1 and from 1 to 0 an infinity of times, without being fixed since then at 0 or 1. However, it is possible to prove that if this value “not fixed” is changed with 0 (or 1), the infinite model of computation will be the same (i.e. we will be able to compute exactly the same functions), the model is slightly less natural.

3 TRANSFINITE STREAM CIPHERS

3.1 One-time pad on finite sets

It is well known that we can design an unconditionally secure encryption scheme on finite sets by using the “one-time pad” algorithm. This algorithm has many nice properties: it is very simple to describe, very fast, and it is proved “unconditionally secure”. By “unconditionally secure”, we mean that if Alice communicates with Bob with the one-time pad algorithm (with all the keys perfectly generated and stored), then we can prove that Charlie will obtain no new information on the cleartexts by listening the encrypted messages, except a maximum value for the length of the cleartexts. This is true even if Charlie has access to arbitrary infinite computing power. The security relies here on information theory, and not on the available computing power. The one-time pad has, however, many inconveniences: all the bits of the secret key must be perfectly random, and each time we encrypt one bit of cleartext we consume one bit of the key, so we will have to regenerate and securely transmit new bits of key. Moreover, the one-time pad algorithm is “malleable” (more details about this property will be given below). None of these inconveniences is so important that it kills the use of the algorithm: there are many variants of the one-time pad that avoid some of these problems, and this algorithm is really used for some very important applications (the “red line” between Moscow and Washington, until recently, for example). These applications are mainly military applications: the one-time pad

algorithm is very simple to describe but delicate and relatively expensive to use correctly.

The “malleability” problem is this one: let assume that Alice encrypts the message M by sending $X = M \oplus K$, where K is the secret key, and \oplus is the bit by bit Xor operation (it is also possible to use any other group operation). If Charlie can obtain the ciphertext, and if Charlie knows, or can guess with a good probability, the cleartext M (such situation can occur), then Charlie is able to find K by $K = X \oplus M$, and now Charlie can encrypt any message of his choice M' by $X' = M' \oplus K$, and send X' to Bob instead of X . This means that from the knowledge of a cleartext/ciphertext pair, Charlie is able to generate another (and even all) possible cleartext/ciphertext pair. This is consistent with the “unconditional security” of the one-time pad, since the “unconditional security” theorem just says that Charlie will not get more information of the cleartext than what he knows by getting the ciphertext (at the exception of his length). However if Charlie knows already the cleartext, the “unconditional security” property says nothing. On order to avoid this malleability problem a lot of solutions exist. The idea is to authenticate the fact that the message comes from Alice. Some solutions are based on error-correcting codes theory. Other solutions are based on information theory security and are designed from random Feistel schemes (cf [11]).

3.2 Generality about the one-time pad on infinite sets

As already mentioned in previous paper (for example [2] or [12]), the one-time pad algorithm can be extended easily to the cases of infinite sets. For example, in order to encrypt a real value m of $[0, 1]$, we will send the ciphertext $x = m + k \bmod 1$, where $k \in_R [0, 1]$. Alternatively, if we denote by $m_1, m_2, \dots, m_i, \dots, i \in \mathbb{N}$, the bits of the message M , and $k_1, k_2, \dots, k_i, \dots, i \in \mathbb{N}$, the bits of the secret key K , then the encryption of M can be X with the bits $x_1, x_2, \dots, x_i, \dots, i \in \mathbb{N}$ such that $\forall i \in \mathbb{N}, x_i = m_i \oplus k_i$. In this example we have $\alpha = \aleph_0$, but this solution can be extended without any difficulty for any cardinal α . We can notice that the one-time pad on infinite sets has a big advantage compared with the one-time pad on finite sets: it is not necessary to generate and communicate securely and regularly new bits of keys after the encryption of some messages (this is true as long as the number of messages to send is $\leq \alpha$, and this is a very natural hypothesis here, since Alice, Bob, and Charlie can perform at maximum α computations). This comes from the fact that for any infinite cardinal α , we have $\alpha^2 = \alpha$. (This can be proved from the Axiom of Choice). For $\alpha = \aleph_0$, a countable union of countable sets is countable. Therefore, from a secret key of \aleph_0 random and

independent bits we can generate \aleph_0^2 (or \aleph_0^3, \aleph_0^4 , etc.) keys with random and independent bits. The computations needed to generate these keys from the initial key are very simple since very simple bijections from \mathbb{N} to \mathbb{N}^2 exist for example. In the sections below we will study how to solve the malleability problem (Section 3.3) with transfinite one-time pad variants and how we can proceed if the ordinal of the number of messages that Alice will want to send to Bob is not known from the beginning (Section 4).

3.3 Solving the malleability problem for the transfinite one-time pad

This Section 3.3 will present the first important new result of this paper. As far as we know, we will present here, for the first time an encryption scheme secure against malleability, in our model of transfinite cryptography. Moreover, the solutions for this problem are not obvious. Here again, we will present our solutions for $\alpha = \aleph_0$. However it will be easy to extend these solutions to any fixed infinite cardinal. A first idea might be to assume that Alice and Bob have chosen \aleph_0 secret-key bits for each possible message m to be send; it will be the authentication of m . However this idea does not work since there are $\mathfrak{C} = 2^{\aleph_0}$ messages of \aleph_0 bits, and \mathfrak{C} is not countable. Solving this problem is not obvious: we will have to use the fact that between these \mathfrak{C} possible cleartexts, only \aleph_0 will be really send, and of course Alice and Bob do not know which one will be send when they have to exchange their secret key.

First solution: with \mathbb{Q} or \mathbb{D}

The sets \mathbb{Q} (rational numbers) and \mathbb{D} (decimal numbers) are countable, so Alice and Bob can have chosen and shared a secret $s(x)$ for each x of \mathbb{Q} (or of \mathbb{D}). $s(x)$ is here a string of \aleph_0 bits, or a real number (we can see it as the same thing). Then to authenticate a real number y , Alice will give to Bob a sequence $s(x_n)$ with $x_n \rightarrow y$ and $x_n \in \mathbb{Q}$ such that from the x_n values, y is the only one accumulation point. We can notice that we use here the property that \mathbb{Q} (or \mathbb{D}) are dense in \mathbb{R} , despite the fact that they are countable. The solution given here is for only one message y , but if we want to authenticate like this \aleph_0 messages y , it is enough to share $\aleph_0^2 = \aleph_0$ values $s^i(x)$ for $i \in \mathbb{N}$.

Second solution: more direct

(This second solution will be in fact about the same solution with other notations. However with these notations the solution will be easier to generalize to any cardinal α).

Alice and Bob share a secret $s(i)$ and a secret $t(i)$ for all $i \in \mathbb{N}$. $s(i)$ and $t(i)$ are here strings of \aleph_0 bits. Now, in order to authenticate a string Y of \aleph_0

bits, $Y = Y_1, Y_2, \dots, Y_i, \dots, i \in \mathbb{N}$ where the Y_i values are bits, Alice will send to Bob the sequences of the values $s(i)$ for all i such that $Y_i = 1$, and the sequence of values $t(i)$, for all i such that $Y_i = 0$, and she will not send the other values $s(i)$ and $t(i)$. Even if Charlie gets all the values send on this line, he will not be able to authenticate a new (i.e. different from Y) string of \aleph_0 bits from the values $s(i)$ and $t(i)$ send. The solution presented here is for only one message Y , however in order to authenticate like this \aleph_0 messages Y , Alice and Bob just have to share $\aleph_0^2 = \aleph_0$ values $s^j(i)$ and $t^j(i)$ for all $i, j \in \mathbb{N}$.

Replay Attack

An eavesdropper who listens to one message and then is able to reboot the protocol (with the same j) and listen to another different message will be able, by rebooting again to authenticate a third message (consisting of the same common bits of the previous messages). It is however easy to avoid this. For example by not accepting any reboot (i.e. j can only strictly increases), or by using a well ordering as we will explain below in section 3.4.

3.4 Well ordering of a set A

Maybe the reader prefers our first solution above from the second solution, due to the replay attack presented above? We believe that this replay attack is easy to avoid by always increasing j when we reboot. However, if the reader is afraid of attacks when j can be the same after a reboot, we will show here that we can proceed for any set A as we did for \mathbb{N} with the first solution. We give here quickly the main ideas for this. First, all set A has a well ordering (this is equivalent to the Axiom of Choice). We can therefore write $A = \{a_i, 0 \leq i < \alpha\}$, where α is the smallest ordinal such that $|\alpha| = A$. Therefore for each $a_i \in A$, $|\{j, i < j < \alpha\}| = A$. Now when we want to sign the message $M = (m_1, m_2, \dots, m_i, \dots)$, $0 \leq i \leq \alpha$, instead of just signing M , we will sign all the $M_i = (m_1, m_2, \dots, m_i)$, $0 \leq i \leq \alpha$, with the solution given above. For this we will not need only $|A|$ bits for the signature, but $|A^2|$. Moreover, since $|A^2| = |A|$ (from the axiom of choice, again for any set A), it changes nothing for the needed number of bits in this signature. Now if we reboot a number of times $< |A|$, we will not be able to sign a new message, since, as seen above, for each $a_i \in A$, $|\{j, i < j < \alpha\}| = A$.

4 AUTO-REGENERATION OF THE KEYS

Let assume that Alice and Bob initially wanted to exchange \aleph_0 messages with the solutions that we have seen, and that these \aleph_0 messages will be organized as a sequence of β values where β is a countable ordinal (for example $\beta = \omega^2$). However, before sending all the messages (for example after sending the $3\omega + 6$ first messages), Alice and Bob realize that it will be more convenient for them to send the messages not as a sequence of β messages, but as a sequence of β' messages where β' is another countable ordinal (for example: $\beta' = \omega^3$). Can they change from one system of ordinal to another system of countable ordinal, i.e. to re-organize the \aleph_0 bits of the secret-keys not already used by performing at maximum \aleph_0 computations? The answer is yes. By using a representation of the ordinals as explained in Section 2, it is easy to re-organize among the ordinal β' the \aleph_0 bits of the secret key not already used.

Remark: The set of countable ordinals is not countable (it has the cardinal \aleph_1 , i.e. the smallest cardinal $> \aleph_0$). So Alice and Bob cannot affect from the beginning \aleph_0 bits of secret key for each countable ordinal β that can be the ordinal of their future sequences of messages, if all these ordinals are possible. However, they can re-organize the ordinal of their sequences of messages \aleph_0 times, as we have seen.

5 TRANSFINITE SECRET-KEY BLOCK ENCRYPTION

We have seen above how Alice and Bob can obtain security with secret-key algorithms for authentication, signatures or encryption, with α computations for any cardinal α . The algorithms that we have seen are secure even if Charlie can perform more than α computations (the security is based on information theory, not on the number, even infinite, of computations to perform). For secret-key encryption, we have seen a solution based on stream ciphering. Is it also possible to find a solution with block encryption? This means for $\alpha = \aleph_0$ for example: can Alice and Bob share a bijection f from $[0, 1]$ to $[0, 1]$ computable in \aleph_0 computations, and such that f will be secure against Charlie if Charlie is limited to \aleph_0 computations? There are $2^{\mathfrak{C}} = \mathfrak{C}$ (functional) from $[0, 1]$ to $[0, 1]$. Such a bijection, if it is random, would therefore require \mathfrak{C} bits of memory, and Alice and Bob do not have \mathfrak{C} bits of memory, if they have to choose a bijection f which is not truly random but will look like random to Charlie. If we use a Feistel design, for example, it is easy to see that the bijective property is not a real problem: from \aleph_0 -pseudo-random functions (i.e.

not distinguishable from truly random functions when we can only perform at most \aleph_0 computations), it is easy to generate \aleph_0 -pseudo-random permutations from $[0, 1]$ to $[0, 1]$. However, how can we generate \aleph_0 -pseudo-random functions from $[0, 1]$ to $[0, 1]$? This problem seems to be related to the problem of generating \aleph_0 -one way functions, that we will see below (in Section 7). It will be an open problem. Therefore, so far, for transfinite secret-key encryption we will recommend stream encryption...

6 PUBLIC-KEY TRANSFINITE SIGNATURES AND AUTHENTICATIONS

A \aleph_0 -one way function will be, by definition, a function that can be computed with \aleph_0 computations, and such that the probability to invert it with \aleph_0 computations is negligible. We can assume, for example, that such a function is from S to S where S is the set of all the strings of \aleph_0 bits. (Similarly for α -one way functions by changing \aleph_0 with α in the definition, for any fixed infinite cardinal α). We can notice that it is not possible to try all the strings with \aleph_0 bits (or with α bits) since $2^{\aleph_0} > \aleph_0$ (and more generally since $2^\alpha > \alpha$ for any cardinal α : this is Cantor's theorem). Does α -one way function exist for any cardinal α , $\alpha \geq \aleph_0$? This question will be studied in Section 7 below. Here we will assume that such functions exist, and then show that this hypothesis will imply that public-key transfinite signature algorithms will exist (and therefore also public-key transfinite authentication algorithms). More precisely, we will see that variants of Lamport signatures schemes will be convenient. (Classical Lamport signature schemes are designed in finite sets, cf [6], [8] [9])

Let f and g be \aleph_0 -one way functions. Alice's secret key will be the values s_i and t_i (each of them of \aleph_0 bits), $i \in \mathbb{N}$, such that $f(s_i) = S_i$, and $g(t_i) = T_i$, $i \in \mathbb{N}$, and Alice's public key will be the sequences of values S_i and T_i , $i \in \mathbb{N}$. Then, to sign a string Y of \aleph_0 bits, $Y = Y_1, Y_2, \dots, Y_i, \dots, i \in \mathbb{N}$, where the Y_i values are bits, Alice will give to Bob, the sequence of s_i values for all i such that $Y_i = 1$, and the sequence of t_i values for all i such that $Y_i = 0$ (and not the other s_i or t_i values). Bob will check for all s_i that he obtained that $f(s_i) = S_i$ and $Y_i = 1$, and for all t_i that he obtained that $g(t_i) = T_i$ and $Y_i = 0$. Charlie can also get these s_i and t_i values. However, from these values, he cannot sign another string Y' different from Y , because f and g are assumed to be \aleph_0 -one way functions. This solution was described above in order to sign only one message Y . However, if they want to sign \aleph_0 messages, Alice and Bob will just have to share $\aleph_0^2 = \aleph_0$ values $s^j(i)$ and

$t^j(i)$ for $i, j \in \mathbb{N}$.

When the messages are finite, in order to avoid large public-keys, many variant of Lamport signatures have been designed, such as the Merkle trees constructions: a pyramidal construction with hash functions. However, when the messages are infinite we do not need these improvements: since $\alpha^2 = \alpha$, the length of the public key will be α bits for α messages, exactly as for one message. (The algorithm presented here for $\alpha = \aleph_0$ can immediately be generalized for any infinite cardinal α).

7 ONE WAY FUNCTIONS

In classical cryptography, i.e. with finite sets, we do not know if there exist or not one-way functions. (Such a proof of existence will imply $P \neq NP$, the most famous open problem in complexity theory). However we have millions of candidates: many relatively simple constructions seem to be one-way, as far as we know. In transfinite cryptography, with our computation model, the situation is rather different: we do not know if α -one way functions exist, but we (the authors of this paper) do not have any candidate either! We tried many different candidates, but so far all of them have failed: we could prove that they were not α -one way. (In an extended version of this paper, we will give more details about the problems that occur here. One of them is to design a function with a combinatorial explosion in the number of solutions, followed by only few final solutions, and this seems to be much more difficult to design in transfinite computations than with finite computations...). Moreover, it may occur that the solution might be different when we use different values α . For example, \mathfrak{C} -one way functions may exist, but not \aleph_0 -one way functions.

8 HASH FUNCTIONS

In traditional (finite) cryptography, hash functions are public functions, they can have a string of any (finite) number of bits as input, and the output is a string (called the hash value) of about 160 or 256 bits with (at least) these properties:

Property 1: Collision resistance:

It must be impossible, from a practical point of view, to find two different inputs with the same output. (Such collisions exist but to get one must require too many computations to be performed to be practical, typically more than 2^{80} for example).

Property 2: “Blob Property”:

We have the possibility to be engaged on a value, without revealing it. From the knowing of $hash(x)$, no useful information on x must be obtained, from a practical point of view.

These concepts can be generalized to transfinite computations. However property 1 and 2 will now be very different. Property 2 is difficult to obtain, because it is related to the concept of α -one way functions. Property 1, nevertheless, will often be easily obtained with bijections. For example, the set D of all the countable subsets of \mathbb{R} (the set of all real numbers) has cardinal \mathfrak{C} , and it is easy to describe a bijection from D to $[0, 1]$ that can be computed with \aleph_0 operations on any fixed input.

9 CONCLUSION

This paper can be seen as an introduction to transfinite cryptography. We have presented the generalizations of the classical concepts of cryptography for infinite computations bounded by a fixed cardinal α . We have presented the previous work on this subject, and some new results. We have also pointed some new open problems. We have seen that simple solutions for transfinite secret-key cryptography exist for authentication, signature and encryption. For public-key transfinite cryptography, it is however more difficult to design some solutions. In [16] a one way function exists in their model of infinite computations, but no public-key encryption, nor public-key signature. In our, more natural, computation model, the situation is in a way the opposite: the existence of α -one way functions is an open problem, but if they exist, then public signature will also exist.

REFERENCES

- [1] Nancy E. Albert. (2008). Review of A^3 and His Algebra. *Cryptologia*, 32(32):189–196.
- [2] Benny Chor and Eyal Kushilevitz. (1989). Secret Sharing over Infinite Domains. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO ’89*, volume 435 of *Lecture Notes in Computer Science*, pages 299–306. Springer-Verlag.
- [3] J.D. Hamkins and A. Lewis. (2000). Infinite time Turing machines. *Journal of Symbolic Logic*, 65(2):567–604.
- [4] Peter Koepke. (2005). Turing Computations on Ordinals. *The Bulletin of Symbolic Logic*, 11(3):377–397.
- [5] Peter Koepke and Martin Koerwien. (2006). Ordinal Computations. *Mathematical Structures in Computer Science.*, 16(5):867–884.
- [6] Leslie Lamport. (1979). Construting Digital Signatures from a one-way function. Technical Report 98, SRI Intl. CSL.

- [7] Boshra Makar. (1980). Transfinite Cryptography. *Cryptologia*, 4(4):230–237.
- [8] Ralph Merkle. (1987). A digital signature based on a conventional encryption function. In Carl pomrance, editor, *Advances in Cryptology – CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer-Verlag.
- [9] Ralph Merkle. (1989). A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer-Verlag.
- [10] Jacques Patarin. Quelques Généralisations Transfinies du Théorème d’Incomplétude de Gödel. Available from the author - An english version is in preparation.
- [11] Jacques Patarin and Paul Camion. (2005). Design of near-optimal pseudorandom functions and pseudorandom permutation in the information-theoric model. *Cryptology ePrint archive: 2005/135: Listing for 2005*.
- [12] Raphael Phan and Serge Vaudenay. On the Impossibility of Strong Encryption over \aleph_0 . Available from the authors.
- [13] Josef Pieprzyk, Hossein Ghodori, Chris Charnes, and Rei Safavi-Naini. (1996). Cryptography Based on Transcendental Numbers. *ACISP*, pages 96–107.
- [14] Apostolos Syropoulos. (2008). “*Hypercomputation*”. Springer-Verlag.
- [15] Philip D. Welch. (2007). Turing Unbound: Transfinite Computation. In *CIE '2007*, volume 4497 of *Lecture Notes in Computer Science*, pages 768–780. Springer-Verlag.
- [16] David Woodruff and Marten van Dijk. (2002). Cryptology in an Unbounded Model. In Lars R. Knudsen, editor, *Advances in Cryptology – EUROCRYPT '2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 149–164. Springer-Verlag.