Fast Contract Signing with Batch Oblivious Transfer

Ľubica Staneková[1,⋆] and Martin Stanek[2,⋆⋆]

[1] Department of Mathematics, Slovak University of Technology
Radlinského 11, 813 68 Bratislava, Slovakia
`ls@math.sk`

[2] Department of Computer Science, Comenius University
Mlynská dolina, 842 48 Bratislava, Slovakia
`stanek@dcs.fmph.uniba.sk`

Abstract. Oblivious transfer protocol is a basic building block of various cryptographic constructions. We propose a novel protocol – batch oblivious transfer. It allows efficient computation of multiple instances of oblivious transfer protocols. We apply this protocol to improve the fast simultaneous contract signing protocol, recently proposed in [11], which gains its speed from computation of time-consuming operations in advance. Using batch oblivious transfer, a better efficiency can be achieved.

1 Introduction

Oblivious transfer is a cryptographic protocol in which one party (usually called sender) transfers one of two strings to the other party (usually called chooser). The transfer should have the following properties: The chooser should obtain the string of his/her choice but not the other one, and the sender should be unable to identify the chooser's choice. Oblivious transfer is used as a key component in many cryptographic applications, such as electronic auctions [12], contract signing [4, 11], and general multiparty secure computations [8]. Many of these and similar applications make intensive use of oblivious transfer. Therefore, efficient implementation of oblivious transfer can improve the overall speed and applicability of various protocols.

Batch variants of various cryptographic constructions are useful for decreasing computational costs. A batch variant of RSA, suitable for fast signature generation or decryption, was proposed by Fiat [5]. Batch verification techniques [1] can be used for efficient proofs of correct decryptions in threshold systems with applications to e-voting and e-auction schemes.

Simultaneous contract signing is a two-party cryptographic protocol, in which two mutually suspicious parties A and B wish to exchange signatures on a contract. Intuitively, a fair exchange of signatures is one that avoids a situation

⋆ supported by APVT 023302
⋆⋆ supported by VEGA 1/0131/03

where A can obtain B's signature while B cannot obtain A's signature and vice-versa. There are two types of contract signing protocols: the ones that use trusted third party either on-line or off-line [6], and protocols without trusted third party [4, 7]. Protocols without trusted third party are based on gradual and verifiable release of information. Hence, if one participant stops the protocol prematurely, both participants have roughly the same computational task in order to find the other participant's signature.

Recently, a contract signing protocol that allows pre-computation of significant part of the most time consuming operations in advance was proposed in [11]. The protocol makes an extensive use of oblivious transfers (its security depends on the security of oblivious transfers) in each protocol run.

Motivation. Oblivious transfer is frequently used in cryptographic protocols. There are many protocols in which a large number of oblivious transfers is employed in a single protocol instance. Therefore, an efficient implementation of oblivious transfer is a natural way to improve the efficiency of such protocols.

Our Contribution. We present a batch RSA oblivious transfer protocol where multiple independent instances of oblivious transfers can be computed efficiently. The security of the protocol is based on RSA assumption, and we prove it in the random oracle model.

We compare actual implementation of batch RSA oblivious transfer protocol with standard RSA oblivious transfer [11], and oblivious transfer based on the computational Diffie-Hellman assumption [13].

We show the usefulness and applicability of our proposal and improve the simultaneous contract signing protocol [11]. The use of batch RSA oblivious transfers instead of pre-computed oblivious transfers leads to more efficient protocol. Both settings were implemented and compared to illustrate exact decrease of computational costs.

Related Work. The efficiency of computing oblivious transfer influences the overall efficiency of many protocols. Our batch RSA oblivious transfer is a modification of the RSA oblivious transfer protocol from [11]. Other constructions of oblivious transfer employ some kind of ElGamal encryption or computational Diffie-Hellman assumption [13].

Similar problem of amortizing the cost of multiple oblivious transfers, based on computational Diffie-Hellman assumption, has been considered by Naor and Pinkas [13]. We compare our approach with their constructions in Sect. 4.

Our security proofs for batch RSA oblivious transfers make use of random oracles. The application of random oracles in the security analysis of cryptographic protocols was introduced by Bellare and Rogaway [2]. Security proofs in a random oracle model substitute a hash function with ideal, truly random function. This approach has been applied to many practical systems, where the ideal function must be instantiated (usually as a cryptographically strong hash function). Recently, an interesting discussion about plausibility of security proofs in the random oracle model appeared in [10].

The paper is structured as follows. Section 2 presents our main result, the batch RSA oblivious transfer, and its implementation. The protocol for contract signing is described in Sect. 3. We analyse an actual implementation of batch RSA oblivious transfer and the savings of computational costs resulting from its application in Sect. 4.

2 Batch Oblivious Transfer

Oblivious Transfer (OT) protocol, more specifically OT_1^2 protocol, allows two parties (sender and chooser) to solve the following problem. The sender has two strings m_0 and m_1 and transfers one of them to the chooser in accordance with the following conditions:

- the chooser selects a particular m_b which he wishes to obtain ($b \in \{0, 1\}$);
- the chooser does learn nothing about m_{1-b};
- the sender does not know which m_b was transferred.

We modify and extend construction of RSA-based OT_1^2 protocol from [11]. Most oblivious transfer protocols employ some kind of ElGamal encryption. This results in increased computational overhead as the chooser must perform at least one modular exponentiation. Using RSA-based oblivious transfer allows to reduce the chooser's complexity, since the public exponent can be made small. Moreover, RSA decryption with distinct private exponents can be implemented efficiently, leading to Batch RSA [5]. We use this idea for further improvement of computational complexity of RSA-based oblivious transfer.

We employ the following notation through the rest of the section. Let $n = p \cdot q$ be an RSA public modulus (i.e. a product of two distinct primes p and q) and let e, d denote public and private exponents, respectively. Let $Z_n = \{0, 1, \ldots, n-1\}$ and let Z_n^* be the set of all numbers from Z_n relatively prime to n. All computations in protocol descriptions are defined over Z_n, the only exception is bitwise xor operation $\oplus$. We will omit stating explicitly that our operations in the paper are mod n whenever it is clear from the context. The hash function H is modelled as a truly random function (random oracle, see [2]) in the security analysis. For simplicity we write $H(a_1, \ldots, a_l)$ for the hash function applied to the concatenation of l-tuple $(a_1, \ldots, a_l)$. Random, uniform selection of x from the set A is denoted by $x \in_R A$.

We assume the sender (S in protocol description) generates the instance of RSA system and the chooser (C) already has a valid public key of the sender (i.e. a pair (n, e)). Moreover, we assume that the length of H output is not shorter than strings m_0 and m_1. Recall, $b \in \{0, 1\}$ denotes the index of string, which the chooser wants to obtain.

2.1 RSA Oblivious Transfer

The RSA oblivious transfer protocol [11] is a modification of the protocol [9]. Since the protocol is executed multiple times a sufficiently long random string R (chosen by sender) is used to distinguish the instances of the protocol.

1. S → C : $C \in_R Z_n^*$
2. C → S : $x' = x^e C^b$, where $x \in_R Z_n$.
3. S → C : R, E_0, E_1,
   where ciphertexts E_0, E_1 of strings m_0, m_1 are computed as follows:
   $$E_0 = H(R, x'^d, 0) \oplus m_0; \qquad E_1 = H(R, (x'C^{-1})^d, 1) \oplus m_1.$$
4. The chooser decrypts m_b from E_b: $m_b = E_b \oplus H(R, x, b)$.

Since the value x' is uniformly distributed in Z_n, the chooser's security is protected in an information-theoretic sense – the sender cannot determine b, even with infinite computational power. The sender's security can be proved in the random oracle model under RSA assumption. The protocol allows pre-computation of value $(C^{-1})^d$, thus allowing efficient implementation of protocols, where multiple instances of oblivious transfer are required.

Remark 1. Roughly the same efficiency can be obtained (without any pre-computation) by generating C^d randomly first and computing C by exponentiation to the short public exponent. This possibility was neglected by the authors of this protocol. Batch oblivious transfer is even more efficient, as we will see later.

2.2 Batch RSA Oblivious Transfer

The main observation regarding efficiency of RSA oblivious transfer is the fact that multiple parallel executions can use distinct private exponents. This allows to reduce computational complexity of sender using techniques of Batch RSA.

We assume that L oblivious transfers should be performed. Let $m_{i,0}$, $m_{i,1}$ (for $0 \le i < L$) be input strings for i-th oblivious transfer. Similarly, $b_0, \ldots, b_{L-1}$ are indices of those strings, which the chooser wants to obtain. The sender selects L distinct small public RSA exponents $e_0, \ldots, e_{L-1}$, each one relatively prime to $(p-1)(q-1)$, and computes corresponding private exponents $d_0, \ldots, d_{L-1}$. For efficient implementation the public exponents must be relatively prime to each other and $e_i = O(\log n)$, for $i = 0, \ldots, L-1$.

The protocol executes L separate instances of oblivious transfer:

1. S → C : $C_0, C_1, \ldots, C_{L-1} \in_R Z_n^*$
2. C → S : $x'_0, x'_1, \ldots, x'_{L-1}$,
   where $x'_i = x_i^{e_i} C_i^{b_i}$ and $x_i \in_R Z_n$, for $i = 0, \ldots, L-1$.
3. S → C : $\{R_i, E_{i,0}, E_{i,1}\}_{0 \le i < L}$,
   where ciphertexts $E_{i,0}$, $E_{i,1}$ of strings $m_{i,0}$, $m_{i,1}$ are computed as follows:
   $$E_{i,0} = H(R_i, (x'_i)^{d_i}, i, 0) \oplus m_{i,0};$$
   $$E_{i,1} = H(R_i, (x'_i C_i^{-1})^{d_i}, i, 1) \oplus m_{i,1}.$$
4. The chooser decrypts $m_{i,b_0}, \ldots, m_{i,b_{L-1}}$ from $E_{i,b_0}, \ldots, E_{i,b_{L-1}}$:
   $$m_{i,b_i} = E_{i,b_i} \oplus H(R_i, x_i, i, b_i), \quad \text{for } i = 0, \ldots, L-1.$$

One can easily check the correctness of the decryption:

$$\begin{aligned} E_{i,b_i} \oplus H(R_i, x_i, i, b_i) &= H(R_i, (x'_i C_i^{-b_i})^{d_i}, i, b_i) \oplus m_{i,b_i} \oplus H(R_i, x_i, i, b_i) \\ &= H(R_i, (x_i^{e_i} C_i^{b_i} C_i^{-b_i})^{d_i}, i, b_i) \oplus m_{i,b_i} \oplus H(R_i, x_i, i, b_i) \\ &= m_{i,b_i} \end{aligned}$$

Security. The chooser's objective is to hide values $b_0, \ldots, b_{L-1}$ from the sender. The values x'_i are uniformly distributed in Z_n. Thus, the sender cannot compute b_i, even with unrestricted computational power – for each transmitted L-tuple $x'_0, \ldots, x'_{L-1}$ and every possible selection of values $b_0, \ldots, b_{L-1}$ there exist suitable choices $x_0, \ldots, x_{L-1} \in Z_n$ (easily computed by the sender himself):

$$x_0 = (x'_i \cdot C_i^{-b_i})^{d_i}, \qquad \ldots, \qquad x_{L-1} = (x'_{L-1} \cdot C_{L-1}^{-b_{L-1}})^{d_{L-1}}.$$

Hence, all combinations of values $b_0, \ldots, b_{L-1}$ are equiprobable and the sender cannot identify the correct one. The chooser's security is protected unconditionally.

The sender's objective is to hide one string from every pair $m_{i,0}$, $m_{i,1}$ (not knowing which one exactly). We prove this security property of the protocol in random oracle model, where the hash function H is modelled as a random function.

We compare the protocol with the ideal implementation (model). The ideal model uses a trusted third party that receives all $m_{i,0}$ and $m_{i,1}$ from the sender and $b_0, \ldots, b_{L-1}$ from the chooser. After obtaining all inputs, the trusted third party sends the chooser m_{i,b_i}, for $0 \leq i < L$. The ideal model hides the values $m_{i,1-b_i}$ perfectly – no adversary substituting the chooser can learn anything about hidden values. The actual protocol should be comparable with the ideal model in the following sense (for extensive study of various definitions of protocol security in the ideal model see [3]):

> For every distribution on the inputs $\{m_{i,0}, m_{i,1}\}_{0 \leq i < L}$ and any probabilistic polynomial adversary A substituting the chooser in the actual protocol there exists a probabilistic polynomial simulator S_A in the ideal model such that outputs of A and S_A are computationally indistinguishable.

Since the ideal model is secure and outputs of A and S_A are indistinguishable, one can conclude that A does not learn more than allowed by security requirements.

The simulator S_A simulates both the sender and adversary A. Therefore, the verb "send" refers to writing data to input or reading data from output of simulated adversary.

1. S_A selects random $C_0, C_1, \ldots, C_{L-1} \in_R Z_n^*$ and sends them to A. It starts to simulate A on this input.

2. A sends values $x'_0, x'_1, \dots, x'_{L-1} \in Z_n$ to S_A. These values can be computed by adversary A in any way (adversary does not need to follow the protocol).
3. S_A selects random strings $\{R_i, E_{i,0}, E_{i,1}\}_{0 \le i < L}$ as "sender's answer" and sends them in response.
4. S_A continues the simulation of A and monitors all its queries to H. All queries have the form of a quadruple (R, x, i, b). We say that the quadruple (R, x, i, b) is valid if $R_i = R$ and $x'_i C_i^{-b} = x^{e_i}$. All queries not containing a valid quadruple are answered at random. If A asks for $H(R, x, i, b)$, where the argument is a valid quadruple, then S_A asks a trusted third party in the ideal model for $m_{i,b}$. The simulator sets $H(R, x, i, b) = E_{i,b} \oplus m_{i,b}$ to allow A to decrypt $E_{i,b}$ correctly. Whatever A outputs, so does S_A.

The distribution of simulated communication with the adversary A is identical to the distribution of real communication between the sender and A. The only exception is the case when A asks for any valid pair of quadruples $H(R, x, i, 0)$ and $H(R, x^*, i, 1)$, for $i \in \{0, \dots, L-1\}$. In this case, the validity of the quadruples implies $x'_i = x^{e_i}$ and $x'_i C_i^{-1} = (x^*)^{e_i}$. It easily follows that $x \cdot (x^*)^{-1}$ is the decryption of C_i:

$$(x \cdot (x^*)^{-1})^{e_i} = x^{e_i} \cdot (x^*)^{-e_i} = x'_i \cdot (x'_i)^{-1} C_i = C_i.$$

The values C_i are chosen randomly by the simulator S_A. Hence, the adversary cannot construct a pair of valid quadruples, assuming the RSA assumption holds. Therefore the output of S_A cannot be distinguished from the output of A in the real communication with the sender.

Remark 2. Random strings R_i are used in the protocol to ensure distinct inputs of H in different invocations of the protocol.

Remark 3. Less direct construction would use triples $(R_i, (x'_i C_i^{-b_i})^{d_i}, b_i)$ instead of quadruples $(R_i, (x'_i C_i^{-b_i})^{d_i}, i, b_i)$. The simulator would determine the correct value of index i by testing validity of all potential triples.

Implementation. The most time-consuming part of the protocol is step 3, where the sender computes $2L$ RSA decryptions. The use of distinct pairs of encryption/decryption exponents enables to apply batch RSA decryption [5]. The sender needs to compute following decryptions in step 3:

$$(x'_i)^{d_i}, (x'_i C_i^{-1})^{d_i}, \qquad \text{for } i = 0, \dots, L-1.$$

Certainly, only one decryption has to be computed for every i, namely $(x'_i)^{d_i}$. This follows from an observation that $(x'_i C_i^{-1})^{d_i} = (x'_i)^{d_i} (C_i^{d_i})^{-1}$, and C_i can be generated from randomly chosen $C_i^{d_i}$ by encrypting it: $(C_i^{d_i})^{e_i}$ (thus having decryption "for free"). Assuming small size of public (encryption) exponents, the computation can be implemented in such a way that L decryptions $(x'_i)^{d_i}$ require time asymptotically proportional to one decryption, see [5]. Notice, that small public exponents yield efficient implementation of the chooser's part of the protocol as well.

3 LS Protocol

The protocol for contract signing from [11] (we call it LS protocol) is based on construction by Even, Goldreich and Lempel [4]. The main difference between these protocols is a criterion when the contract is considered binding (the original protocol uses threshold acceptance).

Protocols for simultaneous contract signing usually consist of two interlaced protocols. Both participants are in symmetric positions – each of them wants to transfer its own signature in exchange for the other participant's signature. Our description includes both exchanges.

Let us denote by $Sig_A(m)$ a digital signature of a message m created by the participant A. The protocol is independent of chosen digital signature algorithm. Let k be a security parameter, e.g. $k = 128$. For the purposes of contract signing a *C-signature* (or $CSig$) of a message m is defined as a triple:

$$CSig_A(m) = (Sig_A(m, R), Sig_A(R, i, 0), Sig_A(R, i, 1)),$$

for arbitrary $i \in \{1, \dots, k\}$ and a random binary string $R \in \{0, 1\}^k$ long enough to avoid collisions among instances of the protocol. A C-signature is (considered) valid if and only if all its parts are formed correctly and have valid signatures.

3.1 The Protocol

Alice and Bob simultaneously transfer C-signatures of contract M. A symmetric encryption (e.g. one-time pad) of message m with a key K is denoted by $\{m\}_K$. We denote by $A \leftrightarrow B : OT_1^2(m_0, m_1)$ the instance of an oblivious transfer protocol with A playing the role of the sender (possessing two strings m_0, m_1), and B playing the role of the chooser (and selecting the string which he wishes to obtain randomly). Alice chooses random $R_A \in \{0, 1\}^k$ and random symmetric keys $K_{A,i,b}$, for $i \in \{1, \dots, k\}$ and $b \in \{0, 1\}$. Similarly, Bob chooses random $R_B \in \{0, 1\}^k$ and random symmetric keys $K_{B,i,b}$, for $i \in \{1, \dots, k\}$ and $b \in \{0, 1\}$. Let k' be the length of symmetric key and i-th bit of key K is denoted by K^i, i.e. $K_{A,i,b} = K^1_{A,i,b}\, K^2_{A,i,b}\, \dots\, K^{k'}_{A,i,b}$, and $K_{B,i,b} = K^1_{B,i,b}\, K^2_{B,i,b}\, \dots\, K^{k'}_{B,i,b}$.

Both participants check the correctness of received data/signatures immediately (as soon as they can be verified). In case of failure, the participant aborts the protocol.

1. (exchange of the first parts of $CSig$)
   $A \to B$: $R_A, Sig_A(M, R_A)$,
   $B \to A$: $R_B, Sig_B(M, R_B)$.
2. (exchange of encrypted parts of $CSig$)
   $A \to B$: $\{Sig_A(R_A, i, b)\}_{K_{A,i,b}}$, for $i = 1, \dots, k$ and $b = 0, 1$,
   $B \to A$: $\{Sig_B(R_B, i, b)\}_{K_{B,i,b}}$, for $i = 1, \dots, k$ and $b = 0, 1$.
3. (opening one half of encryptions)
   $A \leftrightarrow B$: $OT_1^2(K_{A,i,0}, K_{A,i,1})$, for $i = 1, \dots, k$,
   $B \leftrightarrow A$: $OT_1^2(K_{B,i,0}, K_{B,i,1})$, for $i = 1, \dots, k$.

4. (gradual exchange of symmetric keys) For $w = 1, \ldots, k'$:
   $A \to B$: $K^w_{A,1,0}, K^w_{A,1,1}, \ldots, K^w_{A,k,0}, K^w_{A,k,1}$,
   $B \to A$: $K^w_{B,1,0}, K^w_{B,1,1}, \ldots, K^w_{B,k,0}, K^w_{B,k,1}$.

   Transfers are interlaced, so both parties send the pieces in the iteration $w+1$ only when they already received (and verified) the pieces from previous iteration (i.e. w). Alice and Bob check after each iteration that the half of received pieces is equal to the corresponding pieces of the keys obtained via oblivious transfers. They continue the protocol only if the check is successful.

The most computationally demanding task of the protocol is the step 3, where $2k$ oblivious transfers have to be performed. This leaves the room for efficient implementation of the protocol – by employing efficient oblivious transfers, such as our batch oblivious transfer presented in Sect. 2.2.

4 Implementation and Comparison

This section presents actual comparison of oblivious transfer protocols and their impact on efficiency of LS contract signing protocol. All test were implemented in Java and were performed on Pentium II 400 MHz processor.

The Chinese remainder theorem (CRT) is routinely applied to decrease computational cost of RSA decryption. Both RSA-based implementations of oblivious transfer protocols employed CRT. Employing CRT in batch RSA oblivious transfer requires two binary trees for computations mod p and mod q. Results (decryptions) are combined using CRT just like in "standard" RSA.

4.1 Comparing Oblivious Transfer Implementations

We compare implementation of RSA oblivious transfer (Sect. 2.1), batch RSA oblivious transfer (Sect. 2.2), and OT^2_1 protocol proposed by Naor and Pinkas in [13] based on the computational Diffie-Hellman assumption (we denote this protocol NaPi). NaPi computes in subgroup of order r of Z_s, where s is prime and $r \mid s - 1$. For the purpose of our test we choose 160 bit long r. The hash function is instantiated as SHA-1 in the protocols.

The first graph on Fig. 1 shows combined time spent by the sender and the chooser when performing 128 oblivious transfers simultaneously while increasing the length of the RSA modulus n (for RSA-based protocols) or the length of prime s (for NaPi protocol). The second graph presents combined computational time while increasing the number of oblivious transfers computed in parallel. The length of RSA modulus, and the length of prime s is fixed to 1024 bits in this case.

We compare only on-line computations, off-line (pre-computed) parts of protocols are not considered. On-line computation of NaPi protocol requires two modular exponentiations in a subgroup of order r. Since the length of exponents is 160 bits, the protocol is faster than standard RSA oblivious transfer. However, when multiple oblivious transfers should be performed, batch RSA oblivious transfer is even more efficient. Moreover, NaPi protocol requires additional off-line computation (three exponentiations), while batch RSA oblivious transfer does not employ off-line computation.

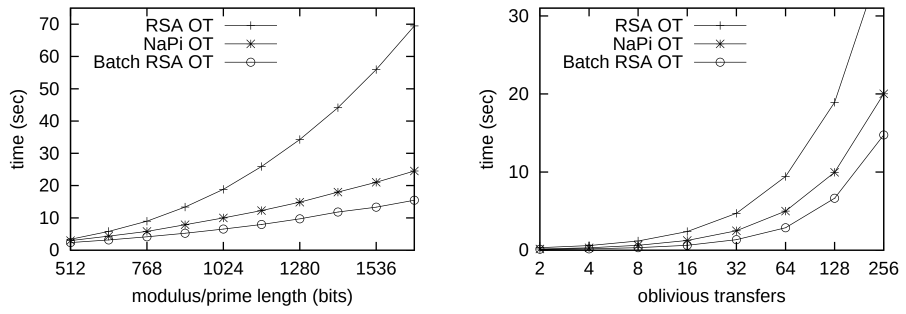

Fig. 1. Comparison of RSA, batch RSA, and NaPi oblivious transfers

Remark 4. Naor and Pinkas proposed additional constructions of oblivious transfer protocols in [13]. They proposed efficient OT_1^N protocol and used it to implement many OT_1^2 protocols using bandwidth/computation tradeoff. However, such construction relies on a fast communication line between the sender and the chooser. Another OT_1^2 protocol proposed by the authors has the advantage of not requiring random oracles for its security proof (and can be viewed as superior to our construction in this sense). On the other hand, its on-line computational complexity is substantially higher.

4.2 Comparing Implementations of LS Protocol

The most time consuming steps of LS protocol are step 2 and step 3. Computational costs of steps 1 and 4 are negligible. Our implementations use RSA modulus of 1024 bits and 128 oblivious transfers (the length of symmetric keys are 128 bits).

Notice the signatures of the second and third parts of C-signatures, i.e. $Sig_A(R_A, i, b)$ and $Sig_B(R_B, i, b)$, do not depend on actual contract M. Thus, they can be pre-computed off-line. Table 1 compares computational time of LS protocol when step 2 is computed on-line (no pre-computation) or off-line (pre-computed signatures). Using batch RSA oblivious transfer improves computational costs in both cases.

Further substantial improvements can be achieved by partitioning keys $K_{A,i,b}$, $K_{B,i,b}$ into larger blocks of length t, e.g. 2 or 3, thus reducing overall number of oblivious transfers by factor t.

Table 1. Computational time of LS protocol (sec)

	on-line	off-line
RSA OT	57.14	19.61
Batch RSA OT	44.25	6.74

References

1. Bellare, M., Garay, J., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures, In *Advances in Cryptology – EuroCrypt '98*, LNCS 1403, 236–250, Springer-Verlag, 1998.
2. Bellare, M., Rogaway, P.: Random Oracles are Practical: a Paradigm for Designing Efficient Protocols, In *1st ACM Conference on Computer and Communication Security*, 62–73, ACM Press, 1993.
3. Canetti, R.: Security and Composition of Multiparty Cryptographic Protocols, *Journal of Cryptology*, Vol. 13, No. 1, 143–202, 2000.
4. Even, S., Goldreich, O., Lempel, A.: A Randomized Protocol for Signing Contracts, In *Advances in Cryptology: Proceedings of Crypto '82*, 205–210, Plenum Publishing, 1982.
5. Fiat, A.: Batch RSA, In *Advances in Cryptology: Proceedings of Crypto '89*, 175–185, LNCS 435, Springer, 1990.
6. Garay, J., Jakobsson, M., MacKenzie, P.: Abuse-Free Optimistic Contract Signing, In *Advances in Cryptology: Proceedings of Crypto '99*, LNCS 1666, 449–466, Springer-Verlag, 1999.
7. Garay, J., Pomerance, C.: Timed Fair Exchange of Standard Signatures, In *Financial Cryptography '03*, LNCS 2742, 190–207, Springer-Verlag, 2003.
8. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game – a completeness theorem for protocols with honest majority, In *19th ACM Symposium on the Theory of Computing*, 218-229, ACM Press, 1987.
9. Juels, A., Szydlo, M.: A Two-Server Sealed-Bid Auction Protocol, In *Financial Cryptography '02*, LNCS 2537, Springer-Verlag, 2002.
10. Koblitz, N., Menezes, A.: Another Look at "Provable Security", Cryptology ePrint Archive, Report 2004/152, http://eprint.iacr.org/, 2004.
11. Liskova, L., Stanek, M.: Efficient Simultaneous Contract Signing, In *19th International Conference on Information Security (SEC 2004)*, 18th IFIP Word Computer Congress, Kluwer Academic Publishers, pp. 441-455, 2004.
12. Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design, In *1st ACM Conference on Electronic Commerce*, 129–139, ACM Press, 1999.
13. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols, In *12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 448–457, 2001.