EFFICIENT SIMULTANEOUS CONTRACT SIGNING

Ľubica Líšková[1] and Martin Stanek[2]

[1] *Department of Mathematics*
Faculty of Civil Engineering
Slovak University of Technology
Radlinského 11, 813 68 Bratislava, Slovak Republic
liskova@math.sk

[2] *Department of Computer Science*
Faculty of Mathematics, Physics and Informatics
Comenius University
Mlynská dolina, 842 48 Bratislava, Slovak Republic
stanek@dcs.fmph.uniba.sk

Abstract Simultaneous contract signing is a two-party cryptographic protocol: two mutually suspicious parties wish to exchange signatures on a contract. We propose novel and efficient protocol for contract signing based on a construction by Even, Goldreich, and Lempel. We focus on the reduction of on-line computational complexity of the protocol. A significant part of the most time-consuming operations can be pre-computed. An important component used in our protocol is an efficient oblivious transfer, which can be of interest per se.

Keywords: contract signing, cryptographic protocols, oblivious transfer

1. INTRODUCTION

Simultaneous contract signing is a two-party cryptographic protocol: two mutually suspicious parties, let us denote them A and B, wish to exchange signatures on a contract. Intuitively, a fair exchange of signatures is one that avoids a situation where A can obtain B's signature while B cannot obtain A's signature and vice-versa. Contract signing protocols can be partitioned into two categories: protocols that use trusted third party either on-line or off-line (so called optimistic protocols) [1, 12, 19], and protocols without trusted third party [5, 9–11, 13]. We are interested in protocols from the second category.

Protocols without trusted third party are based on gradual and verifiable release of information. Hence, if one participant stops the protocol prematurely,

Y. Deswarte et al. (eds.), *Security and Protection in Information Processing Systems*

both participants have roughly the same computational task in order to find the other participant's signature. A security problem can arise in a setting where A has much more hardware power than B. Then the actual time needed for finishing the computation would be unbalanced favorably for A. Timed protocols solve this problem by employing task for which parallel computation has roughly the same complexity as a sequential one [5, 11].

The motivation for our work is the following problem: a server has to participate in many contract signing protocols, possibly with large number of clients. A natural requirement is the time-efficiency of the protocol. An efficient protocol improves the throughput of the server and contributes to the overload protection. A server load is not uniform and thus it makes sense to pre-compute significant part of the protocol beforehand.

Our contribution. We present a contract signing protocol where substantial amount of computation can be done in advance. Our aims are the simplest possible construction and overall efficiency of the protocol. Therefore we based the protocol on construction by Even, Goldreich, and Lempel [10].

A novel oblivious transfer, based on RSA, is used as a key component of the protocol. The security of the contract signing protocol highly depends on the properties of oblivious transfer. We prove the security of oblivious transfer in the random oracle model. To address efficiency issues we show computation vs. communication tradeoff for our oblivious transfer based on ideas from [18].

Let us mention that also ElGamal based oblivious transfer, e.g. [18], can be used in our contract signing protocol, instead of RSA based one. Employing RSA allows further reduction of computation overhead, since public exponent can be made small.

Related work. There are several protocols for contract signing without trusted third party. These protocols have many appealing properties, for example, they are statistical zero-knowledge [9] or resist parallel attack [5, 11, 13], and allow an exchange of standard signatures. On the other hand, strong security properties lead to protocols that employ computationally demanding components, such as zero-knowledge proofs. Although our protocol does not enjoy these properties, it is more efficient than other protocols.

Many cryptographic applications (protocols) make intensive use of oblivious transfer. Hence, efficiency of oblivious transfer influences the overall efficiency of these protocols. Our oblivious transfer is a modification of the protocol by Juels and Szydlo [16]. The problem of amortizing the cost of multiple oblivious transfers has been considered by Naor and Pinkas [18]. They based their construction on computational Diffie-Hellman assumption, while we start with RSA assumption. Both constructions can also be used for reducing the cost of recent protocol for extending few oblivious transfers into many [15].

Our security proofs for 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 [4]. 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, several papers pointed out that the instantiation can break the security of a protocol [3, 7]. On the other hand, the counterarguments are specially designed protocols, unsafe when instantiation takes place.

Organization of the paper. We present and analyze oblivious transfer protocols in Section 2. Section 3 provides an exposition of our contract signing protocol. We evaluate benefits and drawbacks of the protocol in Section 4.

2. OBLIVIOUS TRANSFER

Oblivious Transfer (OT) protocol, more specifically OT_1^N protocol, allows two parties (sender and chooser) to solve the following problem. The sender has N strings $m_0, \ldots, m_{N-1}$ and wishes to transfer one of them to the chooser in a secure manner:

- the chooser can select particular m_b which he wishes to obtain;
- the chooser does not learn any other string except m_b;
- the sender does not know which m_b was transferred.

Oblivious transfer is used as a key component in many cryptographic applications, such as electronic auctions [11, 17], contract signing [10], and general multiparty secure computations [2, 14]. Many of these applications make intensive use of oblivious transfer.

We modify and extend construction of RSA-based OT_1^2 protocol from [16]. 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 special RSA-based oblivious transfer allows to reduce the chooser's complexity. Presented protocols make use of hash function H, modelled as random oracle (truly random function) [4], to facilitate security proofs. First, let us overview the construction from [16].

Notation. We use the following notation through the entire Section 2. All protocols in this paper use RSA public-key system. Let $n = pq$ be an RSA public modulus, where p and q are primes. Let $Z_n = \{0, 1, \ldots, n-1\}$ and $Z_n^* = \{a \in Z_n \mid \gcd(a, n) = 1\}$. An RSA public key consists of modulus n and public exponent e such that $\gcd(e, \varphi(n)) = 1$. Recall that $\varphi(n) = (p-1)(q-1)$ is the Euler function. The corresponding private key (exponent) d satisfies

condition $ed \equiv 1 \pmod{n}$. The protocols involve exclusive knowledge of private key d by the sender. A bitwise exclusive OR operation is denoted by $\oplus$. All other computations are in Z_n. The hash function, denoted by H, is modelled as a truly random function (random oracle) in the security analysis. For simplicity of notation we write $H(a_1, \ldots, a_l)$ for the hash function applied to the concatenation of l-tuple $(a_1, \ldots, a_l)$.

2.1 JS protocol

The protocol assumes that $e = 3$. It proceeds as follows:

1 The sender selects randomly and uniformly an integer $C \in Z_n^*$ as well as keys $k_0, k_1 \in Z_n^*$. Strings m_0, m_1 are encrypted using these keys – resulting in $(H(k_0^3), H(k_0) \oplus m_0)$, $(H(k_1^3), H(k_1) \oplus m_1)$. The sender sends encrypted strings together with C to the chooser.

2 The chooser, wishing to receive m_b for $b \in \{0, 1\}$, selects a random element $x \in Z_n^*$. The chooser computes two values (x_0, x_1):

$$\begin{aligned} (x_0, x_1) &= (x^3, Cx^3) \quad \text{if } b = 0; \\ (x_0, x_1) &= (x^3/C, x^3) \quad \text{if } b = 1; \end{aligned}$$

and sends them to the sender.

3 The sender verifies that $x_1/x_0 = C$ and uses the private key d to compute a reply for the chooser $(z_0, z_1) = (x_0^d k_0, x_1^d k_1)$.

4 The chooser then makes use of x to extract k_b in the obvious fashion: $k_b = z_b/x$. Given k_b, the chooser can extract m_b from $(H(k_b^3), H(k_b) \oplus m_b)$.

The protocol is used as a building block of verifiable proxy oblivious transfer, see [16]. The authors state (without explicit proof) that in the random oracle model the value m_{1-b} is hidden from the chooser in a semantically secure manner, assuming the validity of RSA assumption. The value b is hidden from the sender in an information-theoretic sense.

2.2 Efficient oblivious transfer

The JS protocol can be simplified by observing that x_0 and x_1 always satisfy the equation $x_1/x_0 = C$. Hence sending just one of them is sufficient. We change the relation between x_0 and x_1 to $x_0 = x_1 C$, to free the chooser from the computation of modular inversion. Moreover, we exclude the keys k_0, k_1 from the protocol, since they are not needed.

We want to prove sender's security by comparison with the ideal implementation (model). The ideal model uses a trusted third party that receives m_0

and m_1 from the sender and b from the chooser, and tells the chooser m_b. We require that for every distribution on the inputs (m_0, m_1) and any probabilistic polynomial adversary A substituting the chooser there exists a simulator S_A such that:

1 S_A is a probabilistic polynomial-time machine substituting the chooser in the ideal model;

2 outputs of A and S_A are computationally indistinguishable.

This results in the sender's security, since the ideal model hides the value m_{1-b} perfectly. For extensive study of various definitions of protocol security in the ideal model see [6].

It is not clear how to prove the security of the JS protocol with respect to the ideal model. Therefore we modify the construction of encrypted strings to facilitate the proof.

OT_1^2 protocol. The protocol will run a number of times with some its parameters pre-computed. Therefore we introduce a random string R to differentiate the instances of the protocols.

1 The sender selects randomly and uniformly an integer $C \in Z_n^*$ and sends it to the chooser.

2 The chooser, wishing to receive m_b for $b \in \{0, 1\}$, selects a random element $x \in Z_n$. The chooser computes the value $x' = x^e C^b$ for the sender.

3 The sender selects a sufficiently long random string (or counter) R. Then the sender sets $x_0 = x'$ and computes $x_1 = x_0 C^{-1}$. The sender encrypts strings m_0, m_1 into ciphertexts E_0, E_1:

$$\begin{aligned} E_0 &= H(R, x_0^d, 0) \oplus m_0; \\ E_1 &= H(R, x_1^d, 1) \oplus m_1. \end{aligned}$$

The reply for the chooser consists of values R, E_0, E_1.

4 The chooser then makes use of x to decrypt m_b from E_b: $m_b = E_b \oplus H(R, x, b)$.

The value x' (i.e. x_0) is uniformly distributed in Z_n. For any element x' and any $b \in \{0, 1\}$ there exists an x such that $x' = x^e C^b$. The sender can compute x by $x = (x' \cdot C^{-b})^d$. Hence, 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 is protected computationally. We prove it in the random oracle model, where the hash function H is modelled as a random function. Let A be an adversary. The simulator S_A simulates both the sender and A. First, S_A picks a random value C and sends it to A. When A sends x_0 the simulator computes $x_1 = x_0 C^{-1}$. It selects random strings R, E_0, E_1 and sends them in response (as if they were the sender's answers). S_A continues the simulation of A and monitors all its queries to H. All queries not containing a valid triple (R, x, i), for $i \in \{0, 1\}$, are answered at random. We say that a triple (R', x, i) is valid if $x_i = x^e$ and $R = R'$. If A asks for $H(R, x, i)$, where the argument is a valid triple, then S_A asks a trusted third party in the ideal model for m_i. The simulator sets $H(R, x, i) = E_i \oplus m_i$ to allow A to decrypt correctly. Whatever A outputs, so does S_A.

The distribution of simulated communication with A is identical to the distribution of real communication between the sender and A. Hence the output of S_A cannot be distinguished from the output of A in the real communication with the sender. The only exception is the case when A asks for both valid triples: $H(R, x, 0)$ and $H(R, x^*, 1)$. In this case, from validity of the triples it follows that $x_0 = x^e$ and $x_1 = (x^*)^e$. Then the value $x \cdot (x^*)^{-1}$ is the decryption of C:

$$(x \cdot (x^*)^{-1})^e = x^e \cdot (x^*)^{-e} = x_0 \cdot (x_1)^{-1} = C.$$

Since $C \in Z_n^*$ is random, we cannot distinguish the two distributions with non-negligible probability, assuming the RSA assumption holds.

The input of H contains a random string R to ensure the inputs in different invocations of the protocol are different. We have shown that the knowledge of valid triples is equivalent to breaking of RSA. Hence we can run the protocol with the same value C polynomially many times.

Computation overhead. The protocol allows both parties to pre-compute some values beforehand, since they depend neither on actual values m_0, m_1, b, nor communication between parties. The sender can compute C^{-d} and the chooser can compute x^e. When choosing $e = 3$, the chooser needs two modular multiplications instead of one (expensive) exponentiation. The sender computes x_1^d directly: $x_1^d = x_0^d C^{-d}$, thus performing only single exponentiation x_0^d on-line. Comparison of computation overhead, for $e = 3$ and T instances of the protocol, in a pre-computed and fully on-line implementation, is in Table 1. Notice the saving of exponentiations, which is the most time-consuming operation.

OT_1^N protocol. We extend the OT_1^2 protocol to the case of N strings $m_0, \ldots, m_{N-1}$ employing ideas from [18]. In order to simplify the description of the protocol we set $C_0 = 1$. Although the OT_1^N protocol is not needed for the contract signing protocol directly, it is useful for improving efficiency of the OT_1^2 protocol by means of computation/communication tradeoff. Moreover, this protocol can be of interest per se, since the chooser does not compute any

	on-line		*pre-computed*	
	sender	*chooser*	*sender*	*chooser*
pre-computation				
exponentiations			1	0
multiplications			0	$2T$
on-line				
exponentiations	$2T$	0	T	0
multiplications	T	$3T$	T	T
H evaluations	$2T$	T	$2T$	T

Table 1. OT_1^2 computation overhead: on-line and pre-computed implementation

exponentiation (for small e). The protocol makes use of a random string R (again) to differentiate among multiple invocations.

1 The sender selects randomly and uniformly integers $C_1, \ldots, C_{N-1} \in Z_n^*$ and sends them to the chooser.

2 The chooser, wishing to receive m_b for $0 \leq b < N$, selects a random element $x \in Z_n$. The chooser computes value $x' = x^e C_b$ and sends it to the sender.

3 The sender selects a sufficiently long random string (or counter) R. The sender computes $x_u = x' C_u^{-1}$, for all $0 \leq u < N$, and encrypts strings m_u – resulting in ciphertexts E_u:

$$E_u = H(R, x_u^d, u) \oplus m_u.$$

The reply for the chooser consists of $E_0, \ldots, E_{N-1}$, and R.

4 The chooser decrypts m_b from E_b: $m_b = E_b \oplus H(R, x, b)$.

The protocol has security properties similar to the properties of the OT_1^2 protocol. Since the value x' is uniformly distributed in Z_n, the chooser's security is protected information-theoretically. From the sender's perspective x' can be arbitrary for any b and any $C_1, \ldots, C_{N-1}$ – just set $x = (x' C_b^{-1})^d$.

The sender's security is protected computationally (again). Let A be an adversary. We describe a simulator S_A in the ideal model, using A as a black-box. The simulator and the adversary have computationally indistinguishable outputs in the random oracle model, i.e. assuming a truly random hash function H. The simulator starts with choosing $C_1, \ldots, C_{N-1}$ randomly. S_A "sends" them to A and simulates it to obtain x'. The simulator computes $x_u = x' C_u^{-1}$, for all $0 \leq u < N - 1$. Then, in reply, S_A selects all the values $\{E_u\}_{0 \leq u < N}$ at random. The simulator monitors all queries to (random oracle) H and checks

their validity. We say that the triple (R', x, b) is valid if an only if $x_b = x^e$, $R = R'$ and $0 \leq b < N$. If a triple is not valid, S_A answers the query randomly. In case of valid triple S_A asks a trusted third party in the ideal model for m_b. Then it sets $H(R, x_b, b) = m_b \oplus E_b$ to allow A to decrypt the string correctly. The simulator's output is whatever A outputs.

It is easy to verify that the distribution of simulated communication with A is identical to the distribution of real communication between the sender and A. Hence the output of S_A cannot be distinguished from the output of A in real communication. The only exception is when A queries H with two distinct valid triples, say (R, x, b) and (R, x^*, w), since S_A is unable to get both m_b and m_w from a trusted third party. If A can find two valid triples, s(he) can decrypt the following ciphertext:

$$C_w^{-1} C_b = (x_w (x')^{-1})(x' x_b^{-1}) = x_w x_b^{-1} = (x^* x^{-1})^e.$$

Assuming validity of RSA assumption and the randomness of $C_1, \ldots, C_{N-1}$, the adversary can find two valid triples with negligible probability as long as N is polynomially bounded.

Computation overhead. The Naor-Pinkas OT_1^N protocol from [18] uses computational Diffie-Hellman assumption to guarantee computational security for the sender. Using RSA allows us to reduce chooser's computational overhead, since the public exponent e can be made small.

Similarly to OT_1^2 we can run the protocol polynomially many times with the same values $C_1, \ldots C_{N-1}$. In order to reduce complexity, some of the values can be pre-computed: $C_1^{-d}, \ldots, C_{N-1}^{-d}$ (sender), x^e (chooser). Then the on-line requirements, when running T instances of OT_1^N, are T exponentiations for the sender and no exponentiation for the chooser ($e = 3$).

Remark. The protocol can be easily accommodated in environments where pre-computation is not possible. We sketch briefly a way to reduce the overall number of exponentiations in such cases. The sender selects only t integers $C_0, \ldots, C_{t-1} \in Z_n$, where $t = \lceil \lg N \rceil$. The chooser changes the computation of x' to $x' = x^e C_0^{b_0} \cdot \ldots \cdot C_{t-1}^{b_{t-1}}$, where $(b_0, \ldots, b_{t-1})$ is the binary representation of b. The senders computes $x_u = x' C_0^{-u_0} \cdot \ldots \cdot C_{t-1}^{-u_{t-1}}$, for all $0 \leq u < N$, where $(u_0, \ldots, u_{t-1})$ is the binary representation of u. The security properties of the modified protocol are comparable with the original OT_1^N.

Computation vs. communication tradeoff for OT_1^2. It is quite common that network bandwidth "outperforms" CPU power, i.e. computation is slower than the ability to transfer data over communication lines. The idea of decreasing computation overhead while increasing communication complexity was cleverly used in [18]. The approach can be applied in our protocols as well. The situation is as follows: The chooser and the sender need to perform many

OT_1^2 protocols. They group them in blocks of l protocols (the exact value of l will be specified later). One block is implemented as follows. The sender has l pairs of strings $(m_{1,0}, m_{1,1}), \ldots, (m_{l,0}, m_{l,1})$. Instead of using l instances of the OT_1^2 protocol, the sender creates 2^l strings in the form $m_{1,i_1} m_{2,i_2} \ldots m_{l,i_l}$, where $i_1, \ldots, i_l \in \{0, 1\}$. These strings are the sender's inputs in OT_1^N protocol for $N = 2^l$. The strings can be too long to be encrypted as described in the OT_1^N protocol. Therefore, the sender encrypts them separately and transfer the keys via oblivious transfer. The encryptions are sent off-line.

Optimal performance is achieved when the time spent by computation is equal to the time spent by communication. We evaluate the situation for the sender who performs exponentiation (RSA decryption). Exponentiation is the most computationally intensive operation. Thus, for simplicity, we will ignore other operations. Let t_e (sec) be a time required for one exponentiation. We denote by v (bits/sec) the bandwidth between the chooser and the sender. Communication complexity of the on-line part of the protocol is $2^l k + r$, where r is the length of R and k is the length of encryption key. The optimal value l is then $l = \lg((t_e v - r)/k)$.

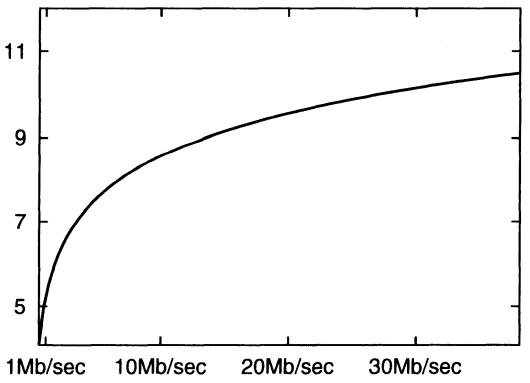

Figure 1. Optimal l for given bandwidth

To show relevance of the described tradeoff, let us give an illustration in the real world settings. Let $k = r = 128$. Let n (RSA modulus) be 1024 bits long. Taking benchmarks from [8], one RSA decryption takes 4.63ms on a 2.1GHz Pentium 4 processor. The optimal value of l as a function of the bandwidth is depicted in Figure 1.

3. CONTRACT SIGNING

The protocol for contract signing from [10] uses an oblivious transfer as a basic component. Our protocol is inspired by this construction but differs from it in the following aspects:

- a criterion when the contract is considered binding (the original protocol uses threshold acceptance);
- an efficient oblivious transfer, described in Section 2.2, allows to compute a significant part of the operations in advance.

These properties guarantee small on-line computation overhead.

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

Let us denote by $Sig_A(m)$ a digital signature of a message m created by the participant A. Our protocol is independent of chosen digital signature algorithm. Let k be a security parameter. For the purposes of contract signing, we define *C-signature* (or $CSig$) of a message m. It is a triple:

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

for arbitrary $i \in \{1, \ldots, k\}$ and a random binary string $R \in \{0,1\}^k$ long enough to avoid collisions among instances of the protocol. A C-signature is 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. We will use A and B as shortcuts for Alice and Bob, respectively. The protocol employs symmetric keys. We assume the length of a symmetric key is a multiple of the security parameter k and not shorter than the length of a signature. Other settings are discussed in Section 3.2. 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.

1 Alice chooses $R_A \in \{0,1\}^k$ and Bob chooses $R_B \in \{0,1\}^k$. Alice and Bob exchange the first parts of their C-signatures together with R_A, R_B:

$A \rightarrow B$: $R_A, Sig_A(M, R_A)$,
$B \rightarrow A$: $R_B, Sig_B(M, R_B)$.

2 Alice and Bob validate the received signatures. They compute the values $Sig_{A/B}(R, i, b)$, for all $i \in \{1, \ldots, k\}$ and $b \in \{0,1\}$. Alice chooses

random symmetric keys $K_{A,i,b}$. Similarly, Bob chooses keys $K_{B,i,b}$. The keys are used for encryption of corresponding signatures. Alice and Bob exchange encrypted signatures, for all $i = 1, \ldots, k$ and $b = 0, 1$:

$A \rightarrow B$: $K_{A,i,b} \oplus Sig_A(R_A, i, b)$,
$B \rightarrow A$: $K_{B,i,b} \oplus Sig_B(R_B, i, b)$.

3 Alice uses an oblivious transfer protocol to exchange exactly one value $K_{A,i,b}$ from each pair of symmetric keys (a pair consists of keys with equal i and different b values):

$A \leftrightarrow B$: $OT_1^2(K_{A,i,0}, K_{A,i,1})$, for $i = 1, \ldots, k$.

Bob selects the key which he wants to receive, the first or the second one for each pair, randomly.

4 Analogously, Bob uses an oblivious transfer protocol to exchange exactly one value $K_{B,i,b}$ from each pair of his keys (Alice's choice for each oblivious transfer is random):

$B \leftrightarrow A$: $OT_1^2(K_{B,i,0}, K_{B,i,1})$, for $i = 1, \ldots, k$.

5 Alice and Bob decrypt half of the signatures using the received keys and validate the signatures. They divide every symmetric key into k pieces of equal length (recall that the length of keys is multiple of k):

$$K_{A,i,b} = K^1_{A,i,b} \,||\, K^2_{A,i,b} \,||\, \ldots \,||\, K^k_{A,i,b},$$
$$K_{B,i,b} = K^1_{B,i,b} \,||\, K^2_{B,i,b} \,||\, \ldots \,||\, K^k_{B,i,b},$$

where the operator '||' denotes concatenation of strings. Alice and Bob gradually exchange the pieces of keys, i.e. for $w = 1, \ldots, k$:

$A \rightarrow B$: $K^w_{A,1,0}, K^w_{A,1,1}, \ldots, K^w_{A,k,0}, K^w_{A,k,1}$,
$B \rightarrow A$: $K^w_{B,1,0}, K^w_{B,1,1}, \ldots, K^w_{B,k,0}, K^w_{B,k,1}$.

Transfers are interlaced, so both parties send the pieces for $w + 1$ only when they already received the pieces for w. Alice and Bob check after each transfer 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.

Security. A valid C-signature consists of three parts. The first part is transferred in step 1. Other parts are exchanged gradually from step 2 to step 5. Security of the protocol heavily depends on the properties of oblivious transfer. They guarantee that Bob learns exactly one part of each signature pair $Sig_A(R_A, i, 0), Sig_A(R_A, i, 1)$ in step 3. Moreover, Alice does not know which of them has been obtained by Bob. Similar situation arises for Bob's signature

pairs in step 4. Hence, neither party knows the C-signature before the last step. Pieces of keys needed to decrypt the remaining signatures are exchanged in step 5. The pieces of known keys are used to check for possible cheating.

The probability of successful cheating in the protocol depends on the security parameter k. Notice that any signature pair completes the C-signature. If a participant wants to cheat (i.e. not providing valid C-signature for the other), it must lie in every pair. Hence the probability of successful cheating is equal to the probability of guessing which keys the other participant obtained in oblivious transfers, i.e. 2^{-k}.

Performance. We analyze on-line and off-line computation overhead of the protocol. Most of the operations can be pre-computed, i.e. computed off-line. It follows from the properties of our OT_1^2 protocol and from the fact that the strings transferred by OT_1^2 do not depend on the contract. As the roles of both participants in the protocol are symmetric, computational and communication needs are equal. We calculate the number of signing operations and exponentiations performed in one instance of the protocol. Other operations, such as signature verification, multiplication, hash function evaluation etc. are neglected in the analysis since they are considerably easier to compute.

Signatures in the first step of the protocol must be computed on-line – they depend on the contract. On the contrary, all of the second step (employing $2k$ signatures) can be pre-computed in advance. Oblivious transfers in steps 3 and 4 require k on-line and 1 off-line exponentiations. The last step involves neither exponentiation nor signing. Table 2 summarizes these facts.

	off-line	*on-line*
signatures	$2k$	1
exponentiations	1	k

Table 2. Computation overhead of a participant

The on-line computation overhead can be further reduced by employing technique of computation/communication tradeoff for OT_1^2 described in Section 2.2.

3.2 Implementation issues

The description of the protocol left open some issues which can be addressed in an actual implementation:

- The security parameter k influences the efficiency of the protocol and the resistance to cheating. An appropriate value of k should balance these requirements, and can be, for example 100 or 128.

- The signatures in step 2 are encrypted using one-time pad. Employing a symmetric encryption algorithm allows signatures much longer than the length of symmetric keys.

- The hash function H is used in OT_1^2 protocol. Although modelled as a truly random function in our analysis, it is instantiated as a cryptographic strong hash function (such as SHA-1 or RIPEMD-160) in the implementation.

- Further simplification removes the hash function H from OT_1^2 protocol altogether. We substitute it with a simple combination of its arguments, for example XOR. This modification breaks the security of OT_1^2 in the random oracle model. On the other hand, we are not aware of any practical attack on our contract signing protocol when employing this modification or modified OT_1^2 protocol itself.

Finally, one can easily change the oblivious transfer protocol in the implementation. It is possible to use ElGamal based oblivious transfer, such as [18]. Nonetheless, employing our RSA based construction with small public exponent e saves exponentiations on the choosers side of the oblivious transfer, see Section 2.2.

4. BENEFITS AND DRAWBACKS

Let us sum up the properties of our contract signing protocol. On the positive side, the protocol is fast and easy to implement. Moreover, it allows any signature algorithm to be used in signing the parts of a C-signature. A precomputation and a computation vs. communication tradeoff can be employed for reducing the on-line complexity of the protocol. We believe the protocol is suitable for a situation where a party (server) has to participate in many contract signing protocols constantly.

When comparing the protocol with other constructions one can spot its drawbacks. Firstly, the protocol does not provide a protection in the case of unbalanced computing power. A participant with greater computing power possesses an advantage over the second one. S(he) can terminate the protocol prematurely and compute the remaining data faster than the other party. This can be reduced by unbalanced release of keys in step 5 if the difference is known in advance.

Additionally, the security proofs for oblivious transfers are based on random oracles. A necessity to instantiate the hash function leaves the possibility of unsafe implementations. On the other hand, we found no unsafe instantiations of the hash function at all.

5. CONCLUSION

We have presented a contract signing protocol with emphasis on performance. We see a possibility for subsequent research in improving the on-line complexity of the protocol further. An attractive objective is to remove the use of random oracles in our RSA based oblivious transfer while preserving its performance.

Acknowledgments

The first author acknowledges partial support from the APVT grant No. 023302. The second author was partially supported by VEGA grant 1/0131/03.

References

[1] N. Asokan, V. Shoup, M. Waidner: Optimistic fair exchange of digital signatures, In *Advances in Cryptology – EuroCrypt '98*, LNCS 1403, 591–606, Springer-Verlag, 2000.

[2] D. Beaver: Minimal-latency secure function evaluation, In *Advances in Cryptology – EuroCrypt '00*, LNCS 1807, 335–350, Springer-Verlag, 2000.

[3] M. Bellare, A. Boldyreva, A. Palacio: An Un-Instantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem, *Cryptology ePrint Archive, Report 2003/077*, Available at `eprint.iacr.org`, 2003.

[4] M. Bellare, P. Rogaway: Random Oracles are Practical: a Paradigm for Designing Efficient Protocols, In *1st ACM Conference on Computer and Communication Security*, 62–73, ACM Press, 1993.

[5] D. Boneh, M. Naor: Timed Commitments, In *Advances in Cryptology – Crypto '00*, LNCS 1880, 236–254, Springer-Verlag, 2000.

[6] R. Canetti: Security and Composition of Multiparty Cryptographic Protocols, *Journal of Cryptology*, Vol. 13, No. 1, 143–202, 2000.

[7] R. Canetti, G. Goldreich, S. Halevi: The Random Oracle Methodology, Revisited, In *Proceedings of the 30th ACM Symposium on Theory of Computing*, 209–218, ACM Press, 1998.

[8] W. Dai: Crypto++ 5.1 Benchmarks, Available at `www.eskimo.com/~weidai/benchmarks.html`.

[9] I.B. Damgård: Practical and provably secure release of a secret and exchange of signatures, In *Advances in Cryptology – EuroCrypt '93*, LNCS 765, 200–217, Springer-Verlag, 1993.

[10] S. Even, O. Goldreich, A. Lempel: A Randomized Protocol for Signing Contracts, In *Advances in Cryptology: Proceedings of Crypto '82*, 205–210, Plenum Publishing, 1982.

[11] J.A. Garay, M. Jakobsson: Timed Release of Standard Digital Signatures, In *Financial Cryptography '02*, LNCS 2537, 168–182, Springer-Verlag, 2002.

[12] J.A. Garay, M. Jakobsson, P.D. MacKenzie: Abuse-Free Optimistic Contract Signing, In *Advances in Cryptology: Proceedings of Crypto '99*, LNCS 1666, 449–466, Springer-Verlag, 1999.

[13] J.A. Garay, C. Pomerance: Timed Fair Exchange of Standard Signatures, In *Financial Cryptography '03*, LNCS 2742, 190–207, Springer-Verlag, 2003.

[14] O. Goldreich, S. Micali, A. Wigderson: 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.

[15] Y. Ishai, J. Kilian, K. Nissim, E. Petrank: Extending Oblivious Transfers Efficiently, *Advances in Cryptology – Crypto 2003*, LNCS 2729, 145–161, Springer-Verlag, 2003.

[16] A. Juels, M. Szydlo: An Two-Server Auction Protocol, In *Financial Cryptography '02*, LNCS 2537, Springer-Verlag, 2002.

[17] M. Naor, B. Pinkas, R. Sumner: Privacy preserving auctions and mechanism design, In *1st ACM Conference on Electronic Commerce*, 129–139, ACM Press, 1999.

[18] M. Naor, B. Pinkas: Efficient oblivious transfer protocols, In *12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 448–457, 2001.

[19] B. Pfitzmann, M. Schunter, M. Waidner: Optimal Efficiency of Optimistic Contract Signing, In *Symposium on Principles of Distributed Computing*, 113-122, 1998.