17. Chaum, D., Pedersen, T.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, Springer, Heidelberg (1993)
18. Chaum, D.: Blind signature system. In: McCurley, K.S., Ziegler, C.D. (eds.) Advances in Cryptology 1981 - 1997. LNCS, vol. 1440, p. 153. Springer, Heidelberg (1999)
19. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994)
20. Cramer, R., Pedersen, T.P.: Improved privacy in wallets with observers (extended abstract). In: Helleseth, T. (ed.) EUROCRYPT 1993. LNCS, vol. 765, pp. 329–343. Springer, Heidelberg (1994)
21. Damgård, I.: Efficient concurrent zero-knowledge in the auxiliary string model. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 418–430. Springer, Heidelberg (2000)
22. Damgård, I., Fujisaki, E.: A statistically-hiding integer commitment scheme based on groups with hidden order. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 125–142. Springer, Heidelberg (2002)
23. George, I.: Davida, Yair Frankel, Yiannis Tsiounis, and Moti Yung. Anonymity control in e-cash systems. In: Financial Cryptography, pp. 1–16 (1997)
24. Jakobsson, M., Yung, M.: Distributed "magic ink" signatures. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 450–464. Springer, Heidelberg (1997)
25. Lipmaa, H.: Statistical zero-knowledge proofs from diophantine equations
26. Nguyen, L.: Accumulators from bilin. pairings and applications. In: Menezes, A.J. (ed.) CT-RSA 2005. LNCS, vol. 3376, Springer, Heidelberg (2005)
27. Stadler, M., Piveteau, J.-M., Camenisch, J.: Fair blind signatures. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 209–219. Springer, Heidelberg (1995)
28. Wei, V.K.: More compact e-cash with efficient coin tracing. Cryptology ePrint Archive, Report 2005/411 (2005) `http://eprint.iacr.org/`

Efficient and Secure Comparison for On-Line Auctions

Ivan Damgård, Martin Geisler, and Mikkel Krøigaard

BRICS, Dept. of Computer Science, University of Aarhus

Abstract. We propose a protocol for secure comparison of integers based on homomorphic encryption. We also propose a homomorphic encryption scheme that can be used in our protocol and makes it more efficient than previous solutions. Our protocol is well-suited for application in on-line auctions, both with respect to functionality and performance. It minimizes the amount of information bidders need to send, and for comparison of 16 bit numbers with security based on 1024 bit RSA (executed by two parties), our implementation takes 0.28 seconds including all computation and communication. Using precomputation, one can save a factor of roughly 10.

1 Introduction

Secure comparison of integers is the problem of designing a two-party or multi-party protocol for deciding whether $n_A \geq n_B$ for given integers n_A, n_B, while keeping n_A, n_B secret. There exists many variants of this problem, depending on whether the comparison result is to be public or not, and whether n_A, n_B are known to particular players, or unknown to everyone. But usually, protocols can be easily adapted to fit any of the variants. Secure comparison protocols are very important ingredients in many potential applications of secure computation. Examples of this include auctions, benchmarking, and secure extraction of statistical data from databases.

As a concrete example to illustrate the application of the results from this paper, we take a closer look at on-line auctions: Many on-line auction systems offer as a service to their customers that one can submit a maximum bid to the system. It is then not necessary to be on-line permanently, the system will automatically bid for you, until you win the auction or your specified maximum is exceeded. We assume in the following what we believe is a realistic scenario, namely that the auction system needs to handle bidders that bid on-line manually, as well as others that use the option of submitting a maximum bid.

Clearly, such a maximum bid is confidential information: both the auction company and other participants in the auction have an interest in knowing such maximum bids in advance, and could exploit such knowledge to their advantage: The auction company could force higher prices (what is known as "shill bidding") and thereby increase its income and other bidders might learn how valuable a given item is to others and change their strategy accordingly.

J. Pieprzyk, H. Ghodosi, and E. Dawson (Eds.): ACISP 2007, LNCS 4586, pp. 416–430, 2007.

In a situation where anyone can place a bid by just connecting to a web site, the security one can obtain by storing the maximum bids with a single trusted party is questionable, in particular if that trusted party is the auction company. Indeed, there are cases known from real auctions where an auction company has been accused of misusing its knowledge of maximum bids.

An obvious solution is to share the responsibility of storing the critical data among several parties, and do the required operations via secure multiparty computation. To keep the communication pattern simple and to minimize problems with maintenance and other logistical problems, it seems better to keep the number of involved players small. We therefore consider the following model:

An input client C supplies an ℓ bit integer m as private input to the computation, which is done by players A and B. Because of our motivating scenario, we require that the input is supplied by sending one message to A, respectively to B, and no further interaction with C is necessary. One may, for instance, think of A as the auction house and B as an accounting company. We will also refer to these as the *server* and *assisting server*.

An integer x (which we think of as the currently highest bid) is public input. As public output, we want to compute one bit that is 1 if $m > x$ and 0 otherwise, i.e., the output tells us if C is still in the game and wants to raise the bid, say by some fixed amount agreed in advance. Of course, we want to do the computation securely so that neither A nor B learns any information on m other than the comparison result.

We will assume that players are honest but curious. We believe this is quite a reasonable assumption in our scenario: C may submit incorrectly formed input, but since the protocol handles even malformed input deterministically, he cannot gain anything from this: any malformed bid will determine a number x_0 such that when the current price reaches x_0, the protocol output will cause C to leave the game. So this is equivalent to submitting x_0 in correct format. Moreover, the actions of A and B can be checked after the auction is over – if C notices that incorrect decisions were taken, he can prove that his bid was not correctly handled. Such "public disgrace" is likely to be enough to discourage cheating in our scenario. Nevertheless, we sketch later in the paper how to obtain active security at moderate extra cost.

1.1 Our Contribution

In this paper, we first propose a new homomorphic cryptosystem that is well suited for our application, this is the topic of Section 2. The cryptosystem is much more efficient than, e.g., Paillier-encryption [1] in terms of en- and decryption time. The efficiency is obtained partly by using a variant of Groth's [2] idea of exploiting subgroups of $\mathbb{Z}_n^*$ for an RSA modulus n, and partly by aiming for a rather small plaintext space, of size $\theta(\ell)$.

In Section 3 we propose a comparison protocol in our model described above, based on additive secret sharing and homomorphic encryption. The protocol is a new variant of an idea originating in a paper by Blake and Kolesnikov [3]. The original idea from [3] was also based on homomorphic encryption but required

a plaintext space of size exponential in ℓ. Here, we present a new technique allowing us to make do with a smaller plaintext space. This means that the exponentiations we do will be with smaller exponents and this improves efficiency. Also, we save computing time by using additive secret sharing as much as possible instead of homomorphic encryption.

As mentioned, our encryption is based on a k bit RSA modulus. In addition there is an "information theoretic" security parameter t involved which is approximately the logarithm of the size of the subgroup of $\mathbb{Z}_n^*$ we use. Here, t needs to be large enough so that exhaustive search for the order of the subgroup and other generic attacks are not feasible. Section 4 contains more information about the security of the protocol.

In the protocol, C sends a single message to A and another to B, both of size $\mathcal{O}(\ell \log \ell + k)$ bits. To do the comparison, there is one message from A to B and one from B to A. The size of each of these messages is $\mathcal{O}(\ell k)$ bits. As for computational complexity, both A and B need to do $\mathcal{O}(\ell(t + \log \ell))$ multiplications mod n. Realistic values of the parameters might be $k = 1024$, $t = 160$, and $\ell = 16$. In this case, counting the actual number of multiplications works out to roughly 7 full scale exponentiations mod n, and takes 0.28 seconds in our implementation, including all computation and communication time. Moreover, most of the work can be done as preprocessing. Using this possibility in the concrete case above, the on-line work for B is about 0.6 exponentiations for A and 0.06 for B, so that we can expect to save a factor of at least 10 compared to the basic version. It is clear that the on-line performance of such a protocol is extremely important: auctions often run up a certain deadline, and bidders in practice sometimes play a strategy where they suddenly submit a much larger bid just before the deadline in the hope of taking other bidders by surprise. In such a scenario, one cannot wait a long time for a comparison protocol to finish.

We emphasize that, while it may seem easier to do secure comparison when one of the input numbers is public, we do this variant only because it comes up naturally in our example scenario. In fact, it is straightforward to modify our protocol to fit related scenarios. For instance, the case where A has a private integer a, B has a private integer b and we want to compare a and b, can be handled with essentially the same cost as in our model. Moreover, at the expense of a factor about 2 in the round, communication and computational complexities, our protocol generalizes to handle comparison of two integers that are shared between A and B, i.e., are unknown to both of them. It is also possible to keep the comparison result secret, i.e., produce it in encrypted form. More details on this are given in Section 5.

Finally, in Section 6 we describe our implementation and the results of a benchmark between our proposed protocol and the one from Fischlin [4].

1.2 Related Work

There is a very large amount of work on secure auctions, which we do not attempt to survey here, as our main concern is secure protocols for comparison, and the on-line auction is mainly a motivating scenario. One may of course do secure

comparison of integers using generic multiparty computation techniques. For the two-party case, the most efficient generic solution is based on Yao-garbled circuits, which were proposed for use in auctions by Naor et al. [5]. Such methods are typically less efficient than ad hoc methods for comparison – although the difference is not very large when considering passive security. For instance, the Yao garbled circuit method requires – in addition to garbling the circuit – that we do an oblivious transfer of a secret key for every bit position of the numbers to compare. This last part is already comparable to the cost of the best known ad hoc methods.

There are several existing ad hoc techniques for comparison, we already mentioned the one from [3] above, a later variant appeared in [6], allowing comparison of two numbers that are unknown to the parties. A completely different technique was proposed earlier by Fischlin in [4].

It should be noted that previous protocols typically are for the model where A has a private number a, B has a number b, and we want to compare a and b. Our model is a bit different, as we have one public number that is to be compared to a number that should be known to neither party, and so has to be shared between them. However, the distinction is not very important: previous protocols can quite easily be transformed to our model, and as mentioned above, our protocol can also handle the other models at marginal extra cost. Therefore the comparison of our solution to previous work can safely ignore the choice of model.

Fischlin's protocol is based on the well-known idea of encrypting bits as quadratic residues and non-residues modulo an RSA modulus, and essentially simulates a Boolean formula that computes the result of the comparison. Compared to [3,6], this saves computing time, since creating such an encryption is much faster than creating a Paillier encryption. However, in order to handle the non-linear operations required in the formula, Fischlin extends the encryption of each bit into a sequence of λ numbers, where λ is a parameter controlling the probability that the protocol returns an incorrect answer. Since these encryptions have to be communicated, we get a communication complexity of $\Omega(\lambda\ell k)$ bits. The parameter λ should be chosen such that $5\ell \cdot 2^{-\lambda}$ is an acceptable (small enough) error probability, so this makes the communication complexity significantly larger than the $\mathcal{O}(\ell k)$ bits one gets in our protocol and the one from [6].

The computational complexity for Fischlin's protocol is $\mathcal{O}(\ell\lambda)$ modular multiplications, which for typical parameter values is much smaller than that of [3,6], namely $\mathcal{O}(\ell k)$ multiplications.[1] Fischlin's result is not directly comparable to ours, since our parameter t is of a different nature than Fischlin's λ: t controls the probability that the best known generic attack breaks our encryption scheme, while λ controls the probability that the protocol gives incorrect results. However, if we assume that parameters are chosen to make the two probabilities be roughly equal, then the two computational complexities are asymptotically the same.

[1] In [3,6] the emphasis is on using the comparison to transfer a piece of data, conditioned on the result of the comparison. For this application, their solution has advantages over Fischlin's, even though the comparison itself is slower.

Thus, in a nutshell, [3,6] has small communication and large computational complexity while [4] is the other way around. In comparison, our contribution allows us to get "the best of both worlds". In Section 6.3 we give results of a comparison between implementations of our own and Fischlin's protocols. Finally, note that our protocol always computes the correct result, whereas Fischlin's has a small error probability.

In concurrent independent work, Garay, Schoemakers and Villegas [7] propose protocols for comparison based on homomorphic encryption that are somewhat related to ours, although they focus on the model where the comparison result is to remain secret. They present a logarithmic round protocol based on emulating a new Boolean circuit for comparison, and they also have a constant round solution. In comparison, we do not consider the possibility of saving computation and communication in return for a larger number of rounds. On the other hand, their constant round solution is based directly on Blake and Kolesnikov's method, i.e., they do not have our optimization that allows us to make do with a smaller plaintext space for the encryption scheme, which means that our constant round protocol is more efficient.

2 Homomorphic Encryption

For our protocol we need a semantically secure and additively homomorphic cryptosystem which we will now describe.

To generate keys, we take as input parameters k, t, and ℓ, where $k > t > \ell$. We first generate a k bit RSA modulus $n = pq$ for primes p, q. This should be done in such a way that there exists another pair of primes u, v, both of which should divide $p-1$ and $q-1$. We will later be doing additions of small numbers in $\mathbb{Z}_u$ where we want to avoid reductions modulo u, but for efficiency we want u to be as small as possible. For these reasons we choose u as the minimal prime greater than $\ell + 2$. The only condition on v is that it is a random t bit prime.

Finally, we choose random elements $g, h \in \mathbb{Z}_n^*$ such that the multiplicative order of h is v modulo p and q, and g has order uv. The public key is now $pk = (n, g, h, u)$ and the secret key is $sk = (p, q, v)$. The plaintext space is $\mathbb{Z}_u$, while the ciphertext space is $\mathbb{Z}_n^*$.

To encrypt $m \in \mathbb{Z}_u$, we choose r as a random $2t$ bit integer, and let the ciphertext be

$$E_{pk}(m, r) = g^m h^r \bmod n.$$

We note that by choosing r as a much larger number than v, we make sure that h^r will be statistically indistinguishable from a uniformly random element in the group generated by h. The owner of the secret key (who knows v) can do it more efficiently by using a random $r \in \mathbb{Z}_v$.

For decryption of a ciphertext c, it turns out that for our main protocol, we will only need to decide whether c encrypts 0 or not. This is easy, since $c^v \bmod n = 1$ if and only if c encrypts 0. This follows from the fact that v is the order of h, uv is the order of g, and $m < u$. If the party doing the decryption has

also stored the factors of n, one can optimize this by instead checking whether $c^v \bmod p = 1$, which will save a factor of 3–4 in practice.

It is also possible to do a "real" decryption by noting that

$$E_{pk}(m, r)^v = (g^v)^m \bmod n.$$

Clearly, g^v has order u, so there is a 1–1 correspondence between values of m and values of $(g^v)^m \bmod n$. Since u is very small, one can simply build a table containing values of $(g^v)^m \bmod n$ and corresponding values of m.

To evaluate the security, there are various attacks to consider: factoring n will be sufficient to break the scheme, so we must assume factoring is hard. Also note that it does not seem easy to compute elements with orders such as g, h unless you know the factors of n, so we implicitly assume here that knowledge of g, h does not help to factor. Note that it is very important that g, h both have the same order modulo both p and q. If g had order uv modulo p but was 1 modulo q, then g would have the correct order modulo n, but $\gcd(g-1, n)$ would immediately give a factor of n. One may also search for the secret key v, and so t needs to be large enough so that exhaustive search for v is not feasible. A more efficient generic attack (which is the best we know of) is to compute $h^R \bmod n$ for many large and random values of R. By the "birthday paradox", we are likely to find values R, R' where $h^R = h^{R'} \bmod n$ after about $2^{t/2}$ attempts. In this case v divides $R - R'$, so generating a few of these differences and computing the greatest common divisor will produce v. Thus, we need to choose t such that $2^{t/2}$ exponentiations is infeasible.

To say something more precise about the required assumption, let G be the group generated by g, and H the group generated by h. We have $H \leq G$ and that a random encryption is simply a uniformly random element in G. The assumption underlying security is now

Conjecture 1. For any constant ℓ and for appropriate choice of t as a function of the security parameter k, the tuple (n, g, h, u, x) is computationally indistinguishable from (n, g, h, u, y), where n, g, h, u are generated by the key generation algorithm sketched above, x is uniform in G and y is uniform in H.

Proposition 2. *Under the above conjecture, the cryptosystem is semantically secure.*

Proof. Consider any polynomial time adversary who sees the public key, chooses a message m and gets an encryption of m, which is of the form $g^m h^r \bmod n$, where g has order uv and h has order v modulo p and q. The conjecture now states that even given the public key, the adversary cannot distinguish between a uniformly random element from H and one from G. But h^r was already statistically indistinguishable from a random element in H, and so it must also be computationally indistinguishable from a random element in G. But this means that the adversary cannot distinguish the entire encryption from a random element of G, and this is equivalent to semantic security – recall that one of the equivalent definitions of semantic security requires that encryptions of m be computationally indistinguishable from random encryptions.

The only reason we set t to be a function of k is that the standard definition of semantic security talks about what happens asymptotically when a *single* security parameter goes to infinity. From the known attacks sketched above, we can choose t to be much smaller than k. Realistic values might be $k = 1024, t = 160$.

A central property of the encryption scheme is that it is homomorphic over u, i.e.,

$$E_{pk}(m, r) \cdot E_{pk}(m', r') \bmod n = E_{pk}(m + m' \bmod u, r + r').$$

The cryptosystem is related to that of Groth [2], in fact ciphertexts in his system also have the form $g^m h^r \bmod n$. The difference lies in the way n, g and h are chosen. In particular, our idea of letting h, g have the same order modulo p and q allows us to improve efficiency by using subgroups of Z_n^* that are even smaller than those from [2].

3 The Protocol

For the protocol, we assume that A has generated a key pair $sk = (p, q, v)$ and $pk = (n, u, g, h)$ for the homomorphic cryptosystem we described previously. The protocol proceeds in two phases: an input sharing phase in which the client must be on-line, and a computation phase where the server and assisting server determine the result while the client is offline.

In the input sharing phase C secret shares his input m between A and B:

- Let the binary representation of m be $m_\ell \dots m_1$, where m_1 is the least significant bit. C chooses, for $i = 1, \dots, \ell$, random pairs $a_i, b_i \in \mathbb{Z}_u$ subject to $m_i = a_i + b_i \bmod u$.
- C sends privately $a_1, \dots, a_\ell$ to A and $b_1, \dots, b_\ell$. This can be done using any secure public-key cryptosystem with security parameter k, and requires communicating $\mathcal{O}(\ell \log \ell + k)$ bits.[2] In practice, a standard SSL connection would probably be used.

In the second phase we wish to determine the result $m > x$ where x is the current public price (with binary representation $x_\ell \dots x_1$).

Assuming a value $y \in \mathbb{Z}_u$ has been shared additively between A and B, as C did it in the first phase, we write $[y]$ for the pairs of shares involved, so $[y]$ stands for "a sharing of" y. Since the secret sharing scheme is linear over $\mathbb{Z}_u$, A and B can compute from $[y]$, $[w]$ and a publicly known value α a sharing $[y + \alpha w \bmod u]$. Note that this does not require interaction but merely local computation. The protocol proceeds as follows:

- A and B compute, for $i = 1, \dots, \ell$ sharings $[w_i]$ where

$$w_i = m_i + x_i - 2x_i m_i = m_i \oplus x_i.$$

[2] We need to send $\ell \log \ell$ bits, and public-key systems typically have $\theta(k)$-bit plaintexts and ciphertexts.

– A and B now compute, for $i = 1, \ldots, \ell$ sharings $[c_i]$ where

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} w_j.$$

Note that if $m > x$, then there is exactly one position i where $c_i = 0$, otherwise no such position exists. Note also, that by the choice of u, it can be seen that no reductions modulo u take place in the above computations.
– Let α_i and β_i be the shares of c_i that A and B have now locally computed. A computes encryptions $E_{pk}(\alpha_i, r_i)$ and sends them all to B.
– B chooses at random $s_i \in \mathbb{Z}_u^*$ and s_i' as a $2t$ bit integer and computes a random encryption of the form

$$\gamma_i = (E_{pk}(\alpha_i, r_i) \cdot g^{\beta_i})^{s_i} \cdot h^{s_i'} \bmod n.$$

Note that, if $c_i = 0$, this will be an essentially random encryption of 0, otherwise it is an essentially random encryption of a random nonzero value. B sends these encryptions to A in randomly permuted order.
– A uses his secret key to decide, as described in the previous section, whether any of the received encryptions contain 0. If this is the case, he outputs "$m > x$", otherwise he outputs "$m \leq x$".

A note on preprocessing: one can observe that the protocol frequently instructs players to compute a number of form $h^r \bmod n$ where r is randomly chosen in some range, typically $[0 \ldots 2^{2t}[$. Since these numbers do not depend on the input, they can be precomputed and stored. As mentioned in the Introduction, this has a major effect on performance because all other exponentiations are done with very small exponents.

4 Security

In this section the protocol is proven secure against an honest but curious adversary corrupting a single player at the start of the protocol.

The client C has as input its maximum bid m and all players have as input the public bid x. The output given to A is the evaluation of $m > x$, and B and C get no output.

In the following we argue correctness and we argue privacy using a simulation argument. This immediately implies that our protocol is secure in Canetti's model for secure function evaluation [8] against a static and passive adversary.

4.1 Correctness

The protocol must terminate with the correct result: $m > x \iff \exists i : c_i = 0$. This follows easily by noting that both $x_i - m_i + 1$ and w_i is nonnegative so

$$c_i = x_i - m_i + 1 + \sum_{j=i+1}^{\ell} w_j = 0 \iff x_i - m_i + 1 = 0 \wedge \sum_{j=i+1}^{\ell} w_j = 0.$$

We can now conclude correctness of the protocol since $x_i - m_i + 1 = 0 \iff m_i > x_i$ and $\sum_{j=i+1}^{\ell} w_j = 0 \iff \forall j > i : m_j = x_j$, which together imply $m > x$. Note that since the sum of the w_j is positive after the first position in which $x_i \neq m_i$, there can be at most one zero among the c_i.

4.2 Privacy

Privacy in our setting means that A learns only the result of the comparison, and B learns nothing new. We can ignore the client as it has the only secret input and already knows the result based on its input.

First assume that A is corrupt, i.e, that A tries to deduce information about the maximum bid based on the messages it sees. From the client, A sees both his own shares $a_1, \ldots, a_\ell$, and the ones for B encrypted under some semantically secure cryptosystem, e.g., SSL. From B, A sees the message:

$$(E_{pk}(\alpha_i, r_i) \cdot g^{\beta_i})^{s_i} \cdot h^{s'_i} \bmod n.$$

By the homomorphic properties of our cryptosystem this can be rewritten as

$$E_{pk}(s_i \cdot \alpha_i, s_i \cdot r_i) \cdot E_{pk}(s_i \cdot \beta_i, s'_i) = E_{pk}(s_i(\alpha_i + \beta_i), s_i \cdot r_i + s'_i).$$

In order to prove that A learns no additional information, we can show that A could – given knowledge of the result, the publicly known number and nothing else – simulate the messages it would receive in a real run of the protocol.

The message received and seen from the client can trivially be simulated as it consists simply of ℓ random numbers modulo u and ℓ encrypted shares. The cryptosystem used for these messages is semantically secure, so the encrypted shares for B can be simulated with encryptions of random numbers.

To simulate the messages received from B, we use our knowledge of the result of the comparison. If the result is "$m > x$", we can construct the second message as $\ell - 1$ encryptions of a nonzero element of $\mathbb{Z}_u^*$ and one encryption of zero in a random place in the sequence. If the result is "$m \leq x$", we instead construct ℓ encryptions of nonzero elements in $\mathbb{Z}_u^*$.

If we look at the encryptions that B would send in a real run of the protocol, we see that the plaintexts are of form $(\alpha_i + \beta_i)s_i \bmod u$. Since s_i is uniformly chosen, these values are random in Z_u if $\alpha_i + \beta_i \neq 0$ and 0 otherwise. Thus the plaintexts are distributed identically to what was simulated above. Furthermore, the ciphertexts are formed by multiplying $g^{(\alpha_i+\beta_i)s_i}$ by

$$h^{s_i r_i + s'_i} = h^{s_i r_i} h^{s'_i}.$$

But h has order v which is t bits long, and therefore taking h to the power of the random $2t$ bit number s'_i will produce something which is statistically indistinguishable from the uniform distribution on the subgroup generated by h. But since $h^{s_i r_i} \in \langle h \rangle$, the product will indistinguishable from the uniform distribution on $\langle h \rangle$. So the s'_i effectively mask out $s_i r_i$ and makes the distribution of the encryption statistically indistinguishable from a random encryption of

$(\alpha_i + \beta_i)s_i$. Therefore, the simulation is statistically indistinguishable from the real protocol messages.

The analysis for the case where B is corrupt is similar. Again we will prove that we can simulate the messages of the protocol. The shares received from the client and the encryptions seen are again simply ℓ random numbers modulo u and ℓ random encryptions and are therefore easy to simulate. Also, B receives the following from A:

$$E_{pk}(\alpha_i, r_i).$$

But since the cryptosystem is semantically secure, we can make our own random encryptions instead and their distribution will be computationally indistinguishable from the one we would get by running the protocol normally.

5 Extensions

Although the protocol and underlying cryptosystem presented in this paper are specialized to one kind of comparison, both may be extended. In this section we will first consider how the protocol can be modified to handle more general comparisons where one input is not publically known, and we will also sketch how active security can be achieved. In the final version of this paper we will consider applications of the cryptosystem to general multiparty computation.

5.1 Both Inputs Are Private

Our protocol extends in straightforward way to the case where A and B have private inputs a, b and we want to compare them. In this case, A can send to B encryptions of the individual bits of a, using his own public key. Since the cryptosystem is homomorphic over u, B can now do the linear operations on the bits of a and b that in the original protocol were done on the additive shares of the bits. Note that B has his own input in cleartext, so the encryptions of the exclusive-or of bits in a and b can be computed without interaction, using the formula $x \oplus y = x + y - 2xy$ which is linear if one of x, y is a known constant. B can therefore produce, just as before, a set of encryptions of either random values or a set that contains a single 0. These are sent to A for decryption. The only extra cost of this protocol compared to the basic variant above is that B must do $\mathcal{O}(l)$ extra modular multiplications, and this is negligible compared to the rest of the protocol.

5.2 Both Inputs Are Shared, Shared Output

The case where both numbers a, b to compare are unknown to A and B can also be handled. Assume both numbers are shared between A and B using additive shares. The only difficulty compared to the original case is the computation of shares in the exclusive-or of bits in a and b. When all bits are unknown to both players, this is no longer a linear operation. But from the formula

$x \oplus y = x + y - 2xy$, it follows that it is sufficient to compute the product of two shared bits securely. Let x, y be bits that are shared so $x = x_a + x_b \bmod u$ and $y = y_a + y_b \bmod u$, where A knows x_a, y_a and B knows x_b, y_b. Now, $xy = x_a y_a + x_b y_b + x_b y_a + x_a y_b$. The two first summands can be computed locally, and for, e.g., $x_a y_b$, A can send to B an encryption $E_{pk}(x_a)$. B chooses $r \in Z_u$ at random and computes an encryption $E_{pk}(x_a y_b - r \bmod u)$ using the homomorphic property. This is sent to A, and after decryption $(x_a y_b - r \bmod u, r)$ forms a random sharing of $x_a y_b$. This allows us to compute a sharing of xy, and hence of $x \oplus y$. Putting this method for computing exclusive-ors together with the original protocol, we can do the comparison at cost roughly twice that of the original protocol.

It follows from an observation in [9] that a protocol comparing shared inputs that gives a public result can always be easily transformed to one that gives the result in shared form so it is unknown to both parties. The basic idea is to first generate a shared random bit $[B]$ where B is unknown to both parties. Then from (bit-wise) shared numbers a, b, we compute two new shared numbers $c = a + (b - a)B, d = b + (a - b)B$, this just requires a linear number of multiplications. Note that $(c, d) = (a, b)$ if $B = 0$ and $(c, d) = (b, a)$ otherwise. Finally, we compare c, d and get a public result B'. The actual result can then be computed in shared form as $[B \oplus B']$.

5.3 Active Security

Finally, we sketch how one could make the protocol secure against active cheating. For this, we equip both A and B with private/public key pairs (sk_A, pk_A) and (sk_B, pk_B) for our cryptosystem. It is important that both key pairs are constructed with the *same* value for u. The client C will now share its input as before, but will in addition send to both players encryptions of all of A's shares under pk_A and all of B's shares under pk_B. Both players are now committed to their shares, and can therefore prove in zero-knowledge during the protocol that they perform correctly. Since the cryptosystem is homomorphic and the secret is stored in the exponent, one can use standard protocols for proving relations among discrete logs, see for instance [10,11,12]. Note that since the two public keys use the same value of u, it is possible to prove relations involving both public keys, for instance, given $E_{pk_A}(x)$ and $E_{pk_B}(y)$, that $x = y$. In the final stage, B must show that a set of values encrypted under pk_A is a permutation of a set of committed values. This is known as the shuffle problem and many efficient solutions for this are known – see, e.g., [13]. Overall, the cost of adding active security will therefore be quite moderate, but the computing the exact cost requires further work: The type of protocol we would use to check players' behavior typically have error probability 1 divided by the smallest prime factor in the order of the group used. This would be $1/u$ in our case, and the protocols will have to be repeated if $1/u$ is not sufficiently small. This results in a tradeoff: we want a small u to make the original passively secure protocol more efficient, but a larger value of u makes the protocols we use for checking players' behavior more efficient. An exact analysis of this is outside the scope of this paper.

6 Complexity and Performance

In this section we measure the performance of our solution through practical tests. The protocol by Fischlin [4] provides a general solution to comparing two secret integers using fewer multiplications than the other known general solutions. We show that in the special case where one integer is publicly known and the other is additively shared between two parties, our solution provides for faster comparisons than our adaptation of [4].

6.1 Setup and Parameters

As described above, our special case consists of a server, an assisting server and a client. The client must be able to send his value and go offline, whereafter the other two parties should be able to do the computations together. In our protocol the client simply sends additive shares to each of the servers and goes offline. However, the protocol by Fischlin needs to be adapted to this scenario before we can make any reasonable comparisons. A very simple way to mimic the additive sharing is for the client to simply send his secret key used for the encoding of his value to the server while sending the actual encoding to the assisting server. Clearly the computations can now be done by the server and assisting server alone, where the server plays the role of the client.

Together, the key and encoding determine the client's secret value, but the key or the encoding alone do not. The key of course reveals no information about the value. Because of semantic security, the encryption alone does not reveal the secret to a computationally bounded adversary.

Another issue is to how to compare the two protocols in a fair way. Naturally, we want to choose the parameters such that the two protocols offer the same security level, but it is not clear what this should mean: some of the parameters in the protocols control events of very different nature. Below, we describe the choices we have made and the consequences of making different choices.

Both protocols use an RSA modulus for their encryption schemes, and it is certainly reasonable to use the same bit length of the modulus in both cases, say 1024 bits. Our encryption scheme also needs a parameter t which we propose to choose as $t = 160$. This is because the best known attack tries to have random results of exponentiations collide in the subgroup with about 2^{160} elements. Assuming the adversary cannot do much more than 2^{40} exponentiations, the collision probability is roughly $2^{2 \cdot 40}/2^{160} = 2^{-80}$.

We do not have this kind of attack against Fischlin, but we do have an error probability of $5\ell \cdot 2^{-\lambda}$ per comparison. If we choose the rationale that the probability of "something going wrong" should be the same in both protocols, we should choose λ such that Fischlin's protocol has an error probability of 2^{-80}. An easy computation shows that for $\ell = 16$, $\lambda = 86$ gives us the desired error probability, and it follows that $\lambda = 87$ works for $\ell = 32$.

We have chosen the parameter values as described above for our implementation, but it is also possible to argue for different choices. One could argue, for instance, that breaking our scheme should be as hard as factoring the (1024

bit) modulus using the best known algorithm, even when the generic attack is used. Based on this, t should probably be around 200. One could also argue that having one comparison fail is not as devastating as having the cryptosystem broken, so that one could perhaps live with a smaller value of λ than what we chose. Fischlin mentions an error probability of 2^{-40} as being acceptable. These questions are very subjective, but fortunately, the complexities of the protocols are linear in t and λ, so it is easy to predict how modified values would affect the performance data we give below. Since we find that our protocol is about 10 times faster, it remains competitive even with $t = 200, \lambda = 40$.

6.2 Implementation

To evaluate the performance of our proposed protocol we implemented it along with the modified version of the protocol by Fischlin [4] described above. The implementation was done in Java 1.5 using the standard BigInteger class for the algebraic calculations and Socket and ServerSocket classes for TCP communication. The result is two sets of programs, each containing a server, an assisting server, and a client. Both implementations weigh in at about 1,300 lines of code. We have naturally tried our best to give equal attention to optimizations in the two implementations.

We tested the implementations using keys of different sizes (k in the range of 512–2048 bits) and different parameters for the plaintext space ($\ell = 16$ and $\ell = 32$). We fixed the security parameters to $t = 160$ and $\lambda = 86$ which, as noted above, should give a comparable level of security.

The tests were conducted on six otherwise idle machines, each equipped with two 3 GHz Intel Xeon CPUs and 1 GiB of RAM. The machines were connected by a high-speed LAN. In a real application the parties would not be located on the same LAN: for credibility the server and assisting server would have to be placed in different locations and under the control of different organizations (e.g., the auction house and the accountant), and the client would connect via a typical Internet connection with a limited upstream bandwidth. Since the client is only involved in the initial sharing of his input, this should not pose a big problem – the majority of network traffic and computations are done between the server and assisting server, who, presumably, have better Internet connections and considerable computing power.

The time complexity is linear in ℓ, so using 16 bit numbers instead of 32 bit numbers cuts the times in half. In many scenarios one will find 16 bit to be enough, considering that most auctions have a minimum required increment for each bid, meaning that the entire range is never used. As an example, eBay require a minimum increment which grows with the size of the maximum bid meaning that there can only be about 450 different bids on items selling for less than $5,000 [16]. The eBay system solves ties by extra small increments, but even when one accounts for them one sees that the 65,536 different prices offered by a 16 bit integer would be enough for the vast majority of cases.

6.3 Benchmark Results

The results of the benchmarks can be found in Tab. 1. From the table it is clear to see that our protocol has performed significantly faster in the tests than the modified Fischlin protocol. The results also substantiate our claim that the time taken by an operation is proportional to the size of ℓ and that we do indeed roughly halve the time taken by reducing the size of ℓ from 32 to 16 bits.

Table 1. Benchmark results. The first column denotes the key size k, the following columns have the average time to a comparison. The average was taken over 500 rounds, after an initial warm-up phase of 10 rounds. All times are in milliseconds. The abbreviation "DGK" refers to our protocol and "F" refers to the modified Fischlin protocol. The subscripts refer to the ℓ parameter used in the timings.

k	DGK_{16}	F_{16}	DGK_{32}	F_{32}
512	82	844	193	1,743
768	168	1,563	331	3,113
1024	280	2,535	544	5,032
1536	564	4,978	1,134	10,135
2048	969	8,238	1,977	16,500

We should note that these results are from a fairly straight-forward implementation of both protocols. Further optimizations can likely be found, in both protocols.

7 Conclusion

This paper has demonstrated a new protocol for comparing a public and a secret integer using only two parties, which among other things has applications in on-line auctions. Our benchmarks suggest that our new protocol is highly competitive and reaches an acceptably low time per comparison for real-world application.

We have also shown how to extend the protocol to the more general case where we have two secret integers and to the active security case. However, further work is needed to evaluate the competitiveness of the extended protocols.

Acknowledgments

The authors would like to thank Tomas Toft, Rune Thorbek, Thomas Mølhave, and the anonymous referees for their comments and suggestions.

References

1. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)