This paper was published in the *Proceedings of the International Workshop on Privacy and Security in Agent-based Collaborative Environments (PSACE)*, 2006, pp. 35–49.

Experimental Evaluation of Security Protocols in SAgent[⋆]

Vandana Gunupudi[1], Stephen R. Tate[2], and Ke Xu[3]

[1] Dept. of Computer Science and Engineering, University of North Texas
gunupudi@cs.unt.edu
[2] Dept. of Computer Science and Engineering, University of North Texas srt@cs.unt.edu
[3] NEC Unified Solutions, Irving, Texas kxu@necunified.com

Abstract. In this paper we report results of experimental exploration of two software methods for protecting the privacy and security of mobile agents while they are operating on untrusted remote hosts. The two methods include a protocol due to Algesheimer, Cachin, Camenisch, and Karjoth (the ACCK protocol) and a protocol due to Tate and Xu (the TX protocol). These protocol implementations are distributed as standard "security providers" in our SAgent security framework, and this paper examines the efficiency of these protocols and explores the trade-off between security and efficiency that is achieved as the different parameters are varied. We believe this is the first experimental testing of these powerful agent protection techniques, and our experimental results show that for simple but useful applications, these techniques provide strong and practical protection for mobile agents.

1 Introduction

Software agents, acting on behalf of the agent originator, must often carry and work with sensitive information while executing on untrusted remote hosts. Protecting the privacy and security of the agents is clearly a fundamental problem which must be addressed before some of the agent applications which have been proposed in the literature are feasible in the real world. In this paper, we present the results of an extensive experimental study of two methods for securing mobile agents, which we refer to as the ACCK protocol [1] and the TX protocol [6]. These two methods are provided as standard protocols in our SAgent security framework, the design of which we describe in a separate paper [3]. In this paper, we focus entirely on the computational aspects: how much time, space, and bandwidth do these methods require? While these methods have been previously described in the theoretical papers, to our knowledge, this is the first experimental exploration.

Protecting the privacy and integrity of sensitive computations in a mobile agent is a nontrivial challenge. Executing on a remote host and having no contact with its originator, the mobile agent is at the full mercy of the hosting environment. Historically, this issue was first approached from a hardware-based perspective, where it was proposed that trusted tamper-proof hardware be installed on systems to host mobile agents [9].

⋆ This research is supported in part by NSF award 0208640.

Software-only solutions which make no trust assumption on the hosting environment, such as the protocols we experiment with in this work, have only recently been proposed [1, 6, 7].

The experimental results presented in this paper show that for simple but useful applications, these techniques provide strong and practical protection for mobile agents. For example, our bid collecting agent, which is capable of maintaining a “best bid” in a completely private manner, executed in under 1 second per visited host with the ACCK protocol using a very secure asymmetric key size of 2048 bits. The TX protocol paid an efficiency penalty by removing the requirement of a trusted third party, but still provided reasonably strong security (with a 768-bit asymmetric key) in under 10 seconds per visited host. It is worth noting that improvements in processor speed that allow larger key sizes to be used only magnify the gap between the agent’s power and an attacker’s power; therefore, if Moore’s Law were to continue for the next 10 years these protocols would be able to handle larger key sizes with good efficiency, but the cost to an attacker would have moved well out of any reasonable reach (assuming no algorithmic improvements for the attackers).

In addition to the overall efficiency measurements, our experiments and presentation in this paper were designed to show how the efficiency is affected by choices in the fundamental parameters provided to these protocols, which gives guidance to SAgent users in making an informed decision regarding the trade-off between security and efficiency.

2 Model

In order to keep security measures as efficient as possible, SAgent-protected applications are decomposed into protected and unprotected computations. For instance, a shopping agent may browse through a store using unprotected computations, but could use SAgent to protect the portions of the agent which make purchasing decisions. The protected portion could include very sensitive information, such as payment authorization values, credit card numbers, or electronic cash tokens. The protected agent computation is modeled as a 3-input, 2-output function, as shown in Figure 1. The three inputs are the current agent state, the input from the host, and an input provided by any unprotected computations performed by the agent on that host. The two outputs are an updated agent state, and an output that is provided to the current host.

The SAgent framework supports arbitrary security *providers*, who supply classes which plug in to the framework according to this model. Specific security providers guarantee slightly different security properties, but in general the agent state is protected so that it is unintelligible to an outside observer (either an attacker or a malicious host), and the host’s input is protected so that it can only be used in a manner consistent with the agent functionality, which can be determined or negotiated as a contract between the originator and the host. In our sample applications, we have experimented with encoding credit card numbers both as constants within the protected computation and as part of the agent state. From an efficiency standpoint there is very little difference, so the application designer is free to choose how this private information is included. As

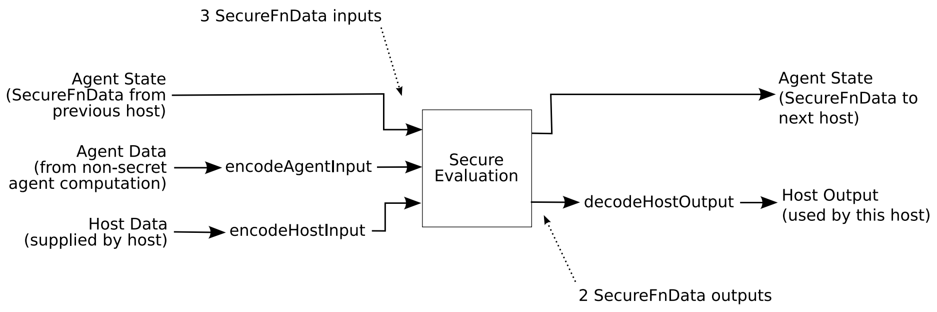

3 SecureFnData inputs
Agent State (SecureFnData from previous host)
Agent Data (from non-secret agent computation)
encodeAgentInput
Host Data (supplied by host)
encodeHostInput
Secure Evaluation
Agent State (SecureFnData to next host)
decodeHostOutput
Host Output (used by this host)
2 SecureFnData outputs

Fig. 1. SAgent Secure Computation Model

an example of an unprotected agent input, one of our applications provides the current time as an input to be used within the protected computation as a timestamp.

3 Secure Agent Computation Protocols

In this section, we give background information on agent protection protocols which form the basis of our current work. The first significant result in software-only protection was obtained by Algesheimer, Cachin, Camenisch, and Karjoth who presented a protocol for secure agent computation of any polynomial-time function [1]. This protocol, while requiring no trusted hardware, involves a third party that is trusted by all agents and hosts. Subsequent work by Tate and Xu further strengthened these results by providing techniques to eliminate this trust assumption [6, 7]. We refer to these two results as the ACCK protocol and the TX protocol, respectively. Both protocols make extensive use of well-established cryptographic results, which boast theoretically provable security but lack practical implementation and performance study. This research intends to fill that gap and provide real-world performance evaluation of these two important theoretical designs.

The ACCK and TX protocols share a common core structure, which we describe next. Building blocks which are different between the two protocols are described in the following sections.

The model of mobile agent computation, as depicted in Figure 1, bears enormous resemblance to the classical two-party Secure Function Evaluation (SFE) problem [8]. Two-party SFE is concerned with evaluating a function by two parties in such a way that the evaluation is carried out honestly and each party learns nothing more than the output it is entitled to. The agent originator and the hosts are the two principals in mobile agents. The agent's purpose is to evaluate, on behalf of the originator, some function at the host with input data from both the host and the agent itself. With two-party SFE, the integrity of the agent computation, as well as the privacy of the agent state, the host input, and the new agent state and host output can all be protected [1, 6, 7]. Next, we describe the 3 basic phases of these secure agent computation protocols.

1. *Initialization:* The mobile agent originator creates an *encrypted circuit* for each sensitive computation to be carried out at a host — the square box in Figure 1. Encrypted circuits are special boolean circuits which compute not on boolean values 0 and 1, but on random strings called "signals" that encode these values. The semantics of the signals, i.e., which strings represent 0 and which strings represent 1, are held and kept secret by the originator. Evaluating an encrypted circuit does not require knowledge of the signals' semantics, nor does evaluation reveal any knowledge about the semantics. Therefore, an encrypted circuit can be evaluated in an untrusted environment without leaking any information about the input, the output, or the intermediate values.

   Throughout the lifetime of a mobile agent, the agent's state is always encoded by strings as just described. The other two inputs, namely the non-secret agent data and the input from the host, also need to be encoded before being fed into the encrypted circuit. For this purpose, the agent carries to the host encrypted versions of all possible input signals. The method used to encrypt these signals is one of the key differences between the ACCK and TX protocols, so we describe these techniques in the following sections.

2. *Evaluation:* Before computing the sensitive agent function, which is in the form of an encrypted circuit, the host gets the signals for its input according to the techniques of either the ACCK or TX protocol. These protocols ensure that only a single input can be retrieved, so malicious hosts cannot reset the agent state and try repeated execution with different inputs (also, if multiple inputs are available, the encrypted circuit no longer has the required privacy properties).

   With the right signals for all three inputs, the encrypted circuit can be evaluated which outputs a new agent state and an output to the host, both in the form of signals. Because of its oblivious nature, the computation is tamper resistant. The originator can include in the agent the semantics of the strings that encodes the host output, so that the host is able to uncover the value of its output by a simple lookup. In contrast, the semantics of the signals for the agent state are never sent out with the agent. Hence, the privacy and integrity of the agent state are always protected.

3. *Finalization:* When the agent returns to the originator, its state will be uncovered by the originator, who holds the semantics for the signals.

With this presentation of the common core of the ACCK and TX protocols, it is clear that the key issue is how signals can be encrypted and included with the agent so that they can be appropriately decrypted when the host and agent need to supply inputs to the protected agent functionality.

3.1 The ACCK Protocol

The ACCK protocol takes a straight-forward approach to signal encryption and decryption based on the assumption that a trusted third party (TTP) is available. The TTP possesses a keypair for some public key algorithm, and makes its public key known to all participants (if necessary, some form of PKI/certificates can be used to ensure authenticity of the keys). The originator then bundles each signal with identifying information and encrypts the bundle using the TTP's public key. In the standard SAgent implementation, RSA is used for this encryption using the Cryptix RSA implementation.

When the agent is executing on a remote host and signals encoding an input are needed, the encrypted versions of the appropriate signals are extracted from the agent and sent to the TTP, who decrypts and verifies (using the bundled identifying information) that only a single input is decrypted for a particular host before returning the decrypted values to the host/agent.

3.2 The TX Protocol

The security of the ACCK protocol requires trust of a third party. In particular, the originator must trust that the TTP will properly enforce the "one input decryption per host/circuit evaluation" rule. The hosts must also trust the TTP to not reveal the signal decryption requests to the originator (otherwise the host's input would be revealed, and it could be used in a manner inconsistent with the agreed-upon behavior of the agent). The TTP therefore becomes a significant trust assumption. The TX protocol was specifically designed to remove reliance on the TTP in the agent computation process.

To eliminate the TTP, the TX protocol makes use of a threshold cryptosystem [5] to encrypt the signals for the host input and the public agent input. The TX protocol requires multiple agents, each carrying a share of the decryption key and visiting a disjoint subset of the hosts. Instead of a trusted third party, the required number of agents (more than the threshold) work together to serve the decryption requests from a host or agent in Step 2 above. However, to truly replace the TTP, we require the additional property that this decryption should not reveal the host's input value to the agents, since the agents represent the originator's interest. This is achieved through a new cryptographic primitive introduced by Tate and Xu, called Oblivious Threshold Decryption (OTD) [7].

OTD combines threshold decryption with the standard cryptographic primitive of oblivious transfer [2] in such a way that the host learns the appropriate decrypted signal value, and the agents which help in the decryption do not learn which input the host has retrieved. More information on how this is accomplished can be found in the original papers [6, 7]. OTD uses discrete-log-based encryption techniques, including the Gennaro-Shoup threshold encryption scheme [5] and the Bellare-Micali oblivious transfer protocol [2], so can be implemented over any cyclic group that supports the computational Diffie-Hellman assumption. Two examples are prime-size subgroups of integers modulo a prime p and a cyclic subgroup of an elliptic curve (EC). SAgent implements OTD using a generic cyclic group (defined using a Java interface), and provides implementations of simple modular and EC groups which can be selected through appropriate parameters in SAgent. While EC-based methods allow use of smaller keys and thus more efficient operation, patents on EC-based encryption should be carefully considered before using this approach in an application.

Oblivious Threshold Decryption enables the TX protocol to forsake the trusted third party and assumes nothing more than that a sufficient number of agents behave honestly.

As agents can be corrupted by hosts, this assumption essentially means that the number of corrupted hosts is limited, and under this assumption the protocol has been rigorously proven secure in the powerful universally composable model of cryptographic security [7].

4 Questions Addressed by Our Experiments

As just described, the ACCK and TX protocols rely on general SFE results, which in turn rely on boolean circuit representation of functions. For complex functions, this is clearly impractical. Therefore, the first question we address in this work is this: Are there interesting and useful functions which are practical when implemented using these protections?

We have run a wide range of experiments with our implementations of these protocols which answer this basic question in the affirmative. We have measured and present in this paper time and space (i.e., agent size) results which demonstrate the effectiveness and practicality of these implementations. In addition, we have also run experiments designed to help us understand how certain key parameters affect the efficiency and resource requirements. In particular, we also explore and present answers to the following questions:

– How does computation time vary with asymmetric key size for the protocols?
– How does computation time vary with *symmetric* key size for the protocols?
– How does the time vary with the size of the agent state?
– How does the time change as the size of the agent or host input is varied?

5 Sample Applications

We developed two sample applications for SAgent, which we describe in this section. Both applications are in the general area of e-commerce agent applications. We use the simpler *Maxbid* application to test the protocols with various keysizes and use the *BuyAgent* application to compare the ACCK and the TX protocols for the typical asymmetric keysize of 1024 bits.

Maxbid application. In this application, the originator wants to sell an item and sends one or more agents out to visit remote hosts to gather offers for the item. Each host gives its bid and the agent keeps the *maximum* bid it has seen so far in its state. We would like to keep the current maximum bid secret so that hosts cannot cheat by bidding slightly over the current maximum. When the agents returns to the originator, the originator decodes each agent's maximum bid and computes the overall maximum bid. The protected function in this application is a comparison of two integer values (current maximum bid and bid offered by current host). In this application there is no unprotected agent input and there is no output to the host.

BuyAgent application. This application is slightly more complex than the *Maxbid* application, using all three of the protected functionality inputs and providing the host with one of the outputs. In this application, the originator wishes to buy an item, so

Legend	Protocol, Group, and Agents
ACCK-1	ACCK with 1 agent
ACCK-3	ACCK with 3 agents
ACCK-4	ACCK with 4 agents
TX-ECC-3	TX with EC-based asymmetric cryptography and 3 agents
TX-ECC-4	TX with EC-based asymmetric cryptography and 4 agents
TX-Zp-3	TX with Z_p-based asymmetric cryptography and 3 agents
TX-Zp-4	TX with Z_p-based asymmetric cryptography and 4 agents

Table 1. Protocol Designation for Graphs

the agent maintains the *minimum* bid it has seen so far. Furthermore, the minimum bid is time-stamped, with the time being provided as the unprotected agent input (unprotected since the current time is not sensitive information). In addition, the originator programs a threshold price called the "buy now" value into the agent, whereby, if any host offers a bid lower than the threshold, the agent provides the host the originator's credit card number to immediately buy the item. Clearly, both the "buy now" value and the credit card number are sensitive pieces of data, and should be protected. The function in this application is a comparison of two integer values, with the evaluation resulting in an update of the minimum value and the corresponding timestamp. The protected function also provides a comparison with the "buy now" threshold, revealing the sensitive information only if the threshold value is met.

6 Environment, Parameters, and Measurements

The experiments were performed on a cluster of seven 2GHz Pentium IV machines, all running Fedora Core 4 Linux. We used Sun's Java SDK 1.5 with JADE version 3.2 and an instrumented version of SAgent 0.9. One machine was designated as the originator and there were 6 visited hosts. For baseline measurements we used a simple SAgent provider called *Insecure*, where the agents simply visited the hosts and performed the required operations but with unprotected techniques. For the ACCK protocol, the number of agents was varied with 1, 3, or 4 agents visiting the 6 remote hosts. For the TX protocol, the threshold property requires at least 3 agents, so we tested the protocol with 3 or 4 agents visiting the remote hosts. Table 1 shows the abbreviations we use on graphs to refer to the basic combinations of protocol, cryptographic base, and number of agents.

6.1 Parameters

Within the SAgent framework's implementation of the ACCK and TX protocols, there are several easily adjusted parameters that affect the behavior of these protocols. The protocols use both symmetric and asymmetric (public key) cryptographic primitives, and the keysize of these encryption schemes can be varied. The symmetric key size can be arbitrarily chosen, and we ran experiments with the key size being 80 bits, 128 bits, and 192 bits. Asymmetric key sizes vary dramatically depending on the algorithm used, or more precisely on the cyclic group that algorithms operate over. The simple modular arithmetic groups, which we refer to generically as Z_p, require much larger key sizes than those based on Elliptic Curve Cryptography, which we refer to generically as ECC. The following table shows comparable key sizes (in bits) between these two classes of algorithms, as specified by the National Institute of Standards and Technology (NIST) and the IEEE standard on Public Key Cryptography [4].

Z_p	768	1024	1536	2048
ECC	128	160	192	224

SAgent supports all of these key sizes with hard-coded cyclic groups (drawn from standard groups when possible), but can use arbitrary cyclic groups if the user specifies the appropriate parameters.

In addition to cryptographic parameters for the protocols, the applications have tunable parameters as well. Both applications work with "bids," which are integer values that can be any number of bits. The *BuyAgent* application includes a timestamp and a credit card number as the agent input and part of the agent state, respectively, so varying the size of these parameters affects the size of the inputs to the encrypted circuits. In our experiments, we varied these parameters by drawing from a fixed set of possibilities, shown in the following table.

Bid (i.e., host input) size (in bits): $\{16, 24, 32\}$
Credit card number size (in bits): $\{16, 24, 32, 64\}$
Timestamp (i.e., agent input) size (in bits): $\{32, 48, 64\}$

6.2 Measurements

We instrumented the standard SAgent distribution in order to measure the following values.

– **Initialization time**: The initialization step involves the originator creating the encrypted circuits, setting the initial agent state, and then sending the agents out to visit the hosts. Creating the encrypted circuits includes encrypting the input signals for each circuit, which dominates the initialization time.
– **Agent Computation time**: This is time the agent spends on remote hosts, measured from when the originator sends the agents out to the time when they return.
– **Per-Host Computation time**: This is the agent computation time, divided by the number of visited host. This metric gives a measure of the average amount of time an agent spends performing computations (and waiting on results from other parties) on each host that it visits.
– **Total Protocol time**: This is the total time from start to finish for each of the protocols. This includes the initialization time as well as the total agent computation time.

For each combination of parameters, we performed each test five times, discarded the min and max values, and averaged the remaining results.

7 Results

The first question we ask is both the simplest and the most important: are these methods practical? The following table shows the total time taken over all stages of the *Maxbid* application (initialization, evaluation, and finalization) by each protocol, using 16-bit bids and a strong symmetric key size of 128 bits (for comparison with these values, the insecure baseline took 1.04 seconds). For the TX protocol, all executions were done with a threshold value of $t = 3$.

Asymm Key	ACCK-1	ACCK-3	ACCK-4	TX-ECC-3	TX-ECC-4	TX-Zp-3	TX-Zp-4
768	4.6	3.55	3.60	55.44	51.49	124.22	107.25
1024	5.72	4.65	4.71	96.44	89.25	273.56	245.89
1536	9.95	8.92	8.89	209.56	185.18	882.05	790.39
2048	17.87	16.97	16.74	385.57	381.71	2101.6	1774.74

These results show that for all reasonable key sizes (i.e., up to 2048 bits), the time required by the ACCK protocol is quite reasonable. For larger key sizes, the TX protocol becomes problematic, but as we discuss later, improvements in processor speed work to our advantage here. Also apparent from this table is that a single agent using the ACCK protocol requires a little more time than when multiple agents visit the remote hosts, since multiple agents can take advantage of the parallelism this affords. However, the improvement going from 3 to 4 agents is not very significant for the ACCK protocol. On the other hand, there is a more noticeable jump in efficiency with the TX protocol changing from 3 to 4 agents, due to the more heavily distributed nature of the TX protocol. When there are only 3 agents, a host requires the cooperation of all 3 agents in order to decrypt its shares, effectively blocking the computations of other hosts. In the case where we have 4 agents, a host can choose any 3 out of the 4 agents as helpers, leading to better load balancing and an improvement in the efficiency of the protocol.

Finally, for the TX protocol, we see a significant improvement in the efficiency, especially for larger key sizes, when we perform cryptographic operations over elliptic curves rather than Z_p. We notice about a five-fold improvement in efficiency when using 224-bit keys over elliptic curves instead of the corresponding 2048-bit key over Z_p.

Next we break down the total time in several ways to see what can be learned about the different phases of the protocols and when considering a more complex application. We also consider space/bandwidth usage.

Initialization times. The time for just the initialization phase of the protocols is reported in Figure 2. These results mirror, on a reduced scale, the total protocol time very closely.

Per Host Computation Times. Figure 3 shows the agent computation times per visited host for the ACCK and the TX protocols, when the time for the initialization phase is removed. As expected, the per host times for the ACCK protocol are very low, with the agent spending less than 1 second at each visited host when using 1024-bit keys. The times are higher for the TX protocol, with the ECC times being significantly faster than those over corresponding Z_p key sizes. For keysizes currently considered secure (1024-bit Z_p and 160-bit ECC), the computation time per visited host when using elliptic curves is only about 13 seconds. In fact, for

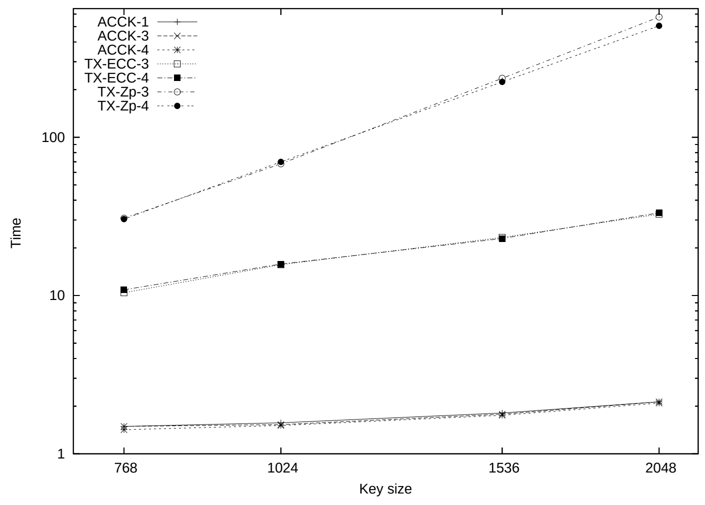


Fig. 2. Initialization time vs. Asymmetric key size (*Maxbid* application)

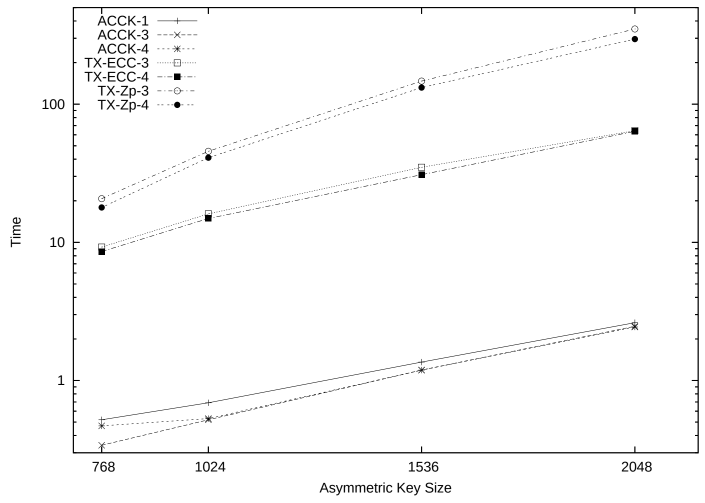
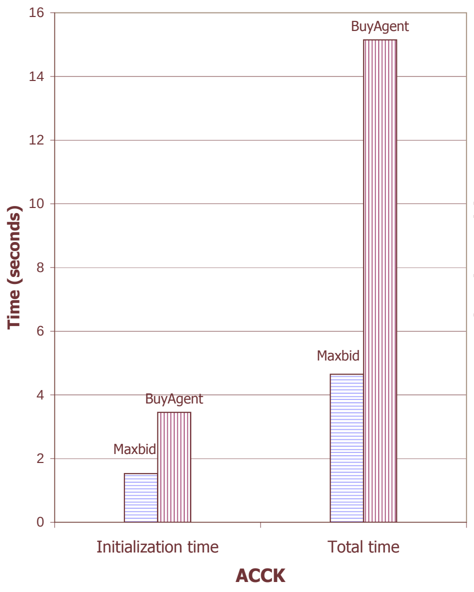


Fig. 3. Per Host Computation Times

keysizes thought to be secure far into the future (2048-bit Z_p and 224-bit ECC), the computation time per visited host is less than 1 minute. Even this is not so bad, considering that, for these larger larger keysizes to be necessary, processor speeds would have to increase by a couple of orders of magnitude, which would mean that these slowest times would clock in at under a second per host.

Efficiency with more complex protected computation. In these tests, we check to see what happens when the complexity of the protected computation increases, by



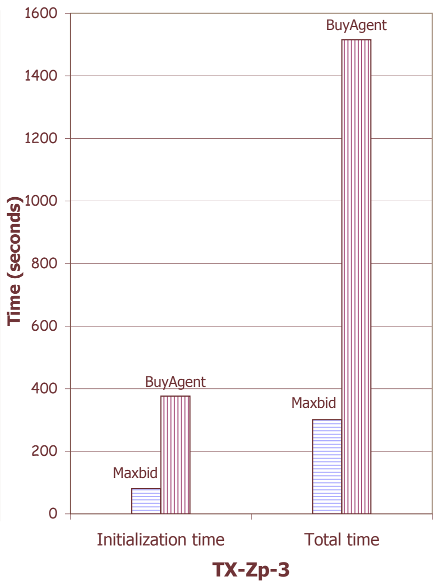


Fig. 4. Comparison of the *Maxbid* and *BuyAgent* applications

changing from the *BuyAgent* application to the *Maxbid* application when the protocol parameters are held constant at 128-bit symmetric keys and 1024-bit asymmetric keys. Both applications are using 16-bit bids, and the *BuyAgent* application is using a 64-bit timestamp and a 64-bit credit card number. Thus the change in applications from *Maxbid* to *BuyAgent* not only increases the complexity of the circuit (from 131 gates to 512 gates), but also increases the agent state size from 16 bits to 144 bits, and increases the agent input from none to 64 bits. The addition of the agent input means that each execution requires 80 signals to be recovered instead of 16, for a five-fold increase. The results are given in Figure 4, showing that the increase in signal decoding requirements is closely mirrored in the increase in total time, verifying that this is the dominant cost.

Space and Bandwidth Usage. Other than time, the amount of space required by the protected agents is an important measure, which also corresponds to the amount of bandwidth required as an agent moves to a new host. The following table shows the initial size of the agents (in kilobytes) for different protocols and symmetric key sizes, with the asymmetric key size set to 1024 bits.

	Maxbid			*BuyAgent*		
	Symmetric Key Size			Symmetric Key Size		
	80	*128*	*192*	*80*	*128*	*192*
ACCK-3	89	97	108	462	506	564
TX-Z_p-3	125	133	145	611	656	717
TX-ECC-3	96	104	116	466	511	572

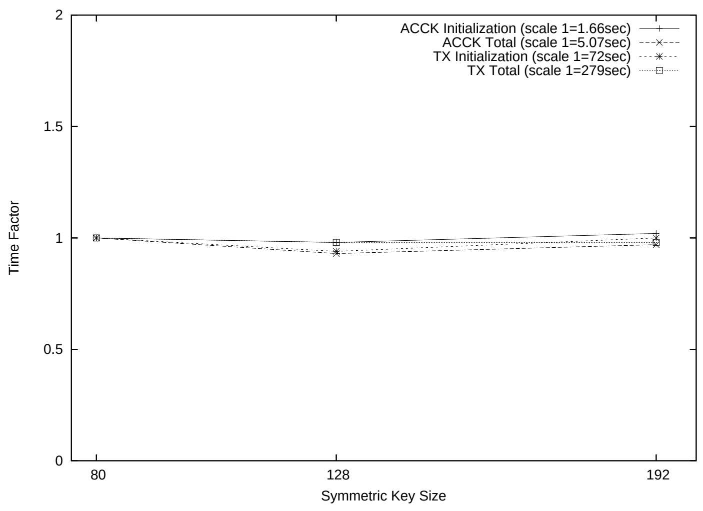


Fig. 5. Variation with symmetric key lengths for the *Maxbid* application

Note that while the 0.9 version of SAgent has been developed with strong security guarantees and time efficiency in mind, it has not been optimized for space usage at all, and we know of places where improvements can be made. Version 1.0 will take this into account, and so the space usage numbers should decrease noticeably. Still, from these tests with version 0.9 we see that the overall sizes are reasonable for current environments (both system and network), particularly with the ACCK protocol. We also see that while the symmetric key size does not noticeably affect the time, it does affect the agent size, so should be a design consideration for an SAgent user.

Next we examine how efficiency is affected by the other tunable parameters. In these tests we are less concerned with absolute times, and more concerned with changes in time caused by varying parameters. Because of this, to accommodate sets of times with very different ranges on the same graph, we normalize each test so that the initial measurement is always scaled to 1, and other measurements are represented as a factor of this value. In the graphs, the scaling factor for each curve is shown in the graph key so that absolute time values can be derived from these graphs as well.

Variation in Symmetric key sizes. Figure 5 shows the results of varying the symmetric key size (corresponding to the signal sizes for the encrypted circuits), showing that the symmetric key size has very little effect on the overall time. Since the symmetric key sizes are small and always fit in a single encryption block of the asymmetric algorithm, the time of the asymmetric operations (the dominant time of the protocol) is unaffected by symmetric key size. This suggests that the largest symmetric key size could be chosen without any time penalty, although for space savings there doesn't seem to be much need to go above 128 bits.

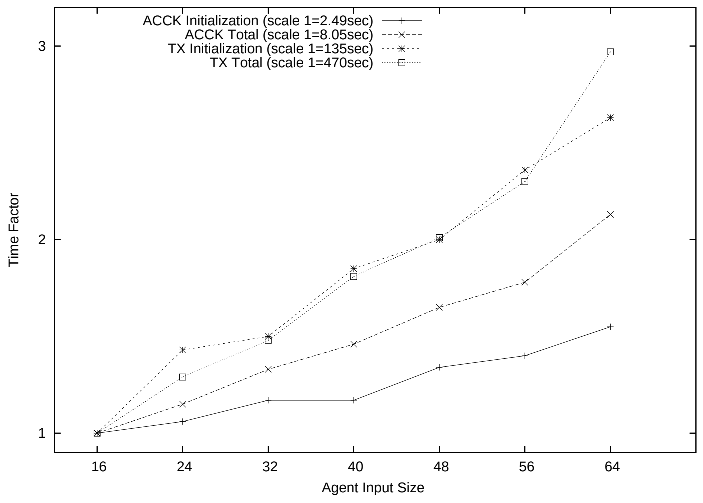


Fig. 6. Variation with agent input size for the *BuyAgent* application

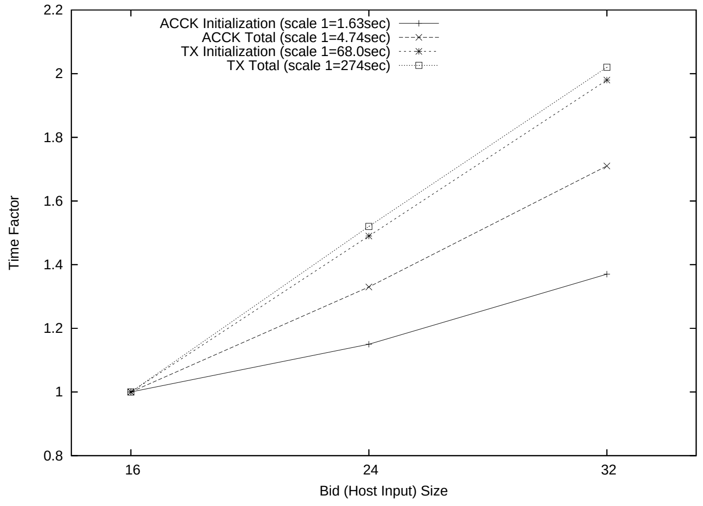


Fig. 7. Variation with bid (host input size) for *Maxbid*

Variation of input sizes. Figure 6 shows how variation of the agent input size increases the time for the *BuyAgent* application. Figure 7 shows the variation of the host input size (i.e., the bid size) for the *Maxbid* application. For both the protocols, the total protocol times as well as the initialization times show a steady increase that is roughly linear, reflecting the increased time to get the decryption of the additional input bits. For the ACCK protocol the cost is approximately 160ms of total protocol time per bit, and for the TX protocol the cost is approximately 15 seconds per bit. The TX cost per extra bit is significant, leading to the conclusion that for TX an input encoding should be chosen that keeps this in mind and uses the absolute fewest possible bits.

Variation of credit card size for *BuyAgent*. Recall that the *BuyAgent* application carries a credit card number as secret data within the agent, which is revealed only if the bid from the host is less than the (also secret) “buy now” value carried by the agent. We allow the credit card number to vary in size (and in fact it could be something quite different like an e-cash token), and show the results of varying this size in Figure 8. The credit card number is actually carried as part of the encoded agent state, so is one of the inputs to the encrypted circuit. However, the results of our test show that the size of this value has very little effect in the overall time. This demonstrates the difference between the agent state input and the other inputs to the encrypted circuit — specifically, no signal decryption or involvement of an external party is required in order to use the bits of the agent state, so variations in the agent state size are much less significant than variations in the other inputs.

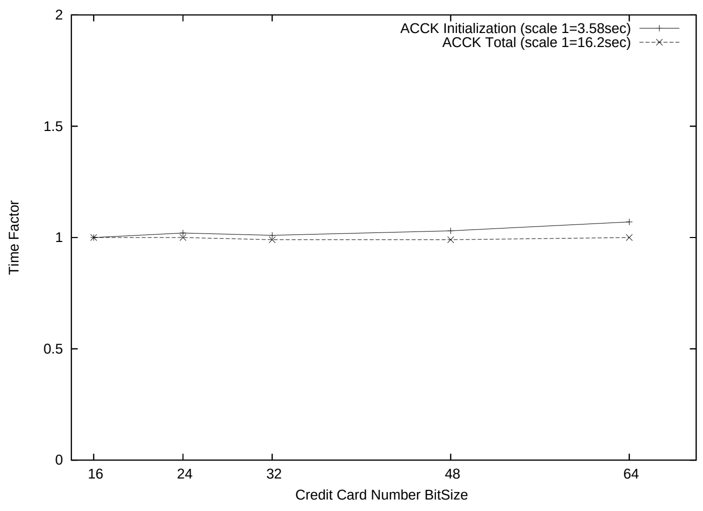


Fig. 8. Variation with credit card number size for *BuyAgent*

The results described in this section allow a potential user of the SAgent security framework to weigh the issues and predict performance of the ACCK and TX protocols. The results show the importance of choosing efficient agent input and host input representations, and the importance of selecting an asymmetric key size that is only as strong as the desired level of security. On the other hand, our experiments have also shown that the symmetric key size and agent state size have very little effect on the overall time of the protocols, so provide the designer some slack in the choice of these parameters.

8 Conclusion

In this paper, we have done extensive experiments to understand the practical performance of two secure agent computation protocols, which protect both the confidentiality and integrity of the agent data. Results show that these protocols are practical for simple but useful applications with reasonable levels of security. While some of the tests using higher security settings and for the more complex application could be considered only borderline secure now, future increases in processor speed will only make these techniques more practical. This is because the key parameter that affects efficiency is the asymmetric key size, and as these key sizes grow the gap between our power and an attacker's only grows greater. For example, if Moore's Law continues for the next 10 years, then processors will be approximately 100 times faster, and 2048-bit keys in the TX-Z_p-4 protocol would result in a reasonable execution time of around 2 seconds per host. And yet, even with 100 times faster processors, without significant algorithmic advances, breaking a 2048-bit key would require many millenia on even huge clusters of these fast machines. Our conclusion is that some software-only protections are possible now, and the future looks promising indeed for these techniques.

References

1. J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 2–11, 2001.
2. M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In G. Brassard, editor, *Advances in Cryptology — Crypto'89*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557. Springer, 1989.
3. V. Gunupudi and S. R. Tate. Sagent: A security framework for JADE. In *Proceedings of AAMAS '06*, 2006.
4. IEEE standard on public key cryptography. http://grouper.ieee.org/groups/1363/, 2000.
5. V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *Journal of Cryptology*, 15:75–96, 2002.
6. S. R. Tate and K. Xu. Mobile agent security through multi-agent cryptographic protocols. In *Proc. of the 4th International Conference on Internet Computing (IC 2003)*, pages 462–468, 2003.
7. K. Xu and S. R. Tate. Universally composable secure mobile agent computation. In *Proceedings of the 7th International Conference on Information Security (ISC)*, pages 304–317, 2004.
8. A. Yao. How to generate and exchange secrets. *Proc. of the 27th IEEE Symposium on Foundations of Computer Science(FOCS)*, pages 162–167, 1986.
9. B. Yee. A sanctuary for mobile agents. *Secure Internet Programming, Lecture Notes in Computer Science*, 1603:261–273, 1999.