On the Concept of Trusted Computing and Software Watermarking: A Computational Complexity Treatise

Wu Hao and Wu Guoqing
School of computer, Wuhan University, Wuhan University, Wuhan, China

Abstract

Trusted computing is a new requirement of information security, after the successful modern cryptography. Although there are many developments of methods and techniques to address on the problems of trusted computing, the lack of rigorous theoretic treatment makes it hard to give further mathematical treatment. In this paper, we develop theory and concepts of trusted computing founded on computational complexity, which analogies to modern cryptography and find close connections between trusted computing and one way functions, one of the pillars of the modern cryptography. After that, we reveal that trusted computing has a subtle but deep relation with software watermarking, that concept can be found on the concept of trusted computing.

Keywords: Trusted computing; Trust measurement; one way function; software watermarking

1. Introduction

Trusted computing is a new requirement of information security, after the successful modern cryptography [1]. But the concept of trusted computing arise from different demands of information system, reliability, robustness, security sometimes share same meaning of trusted computing , which needs a rigor classification according to different requirements and contexts ,for example, in the context of Software Engineering ,Trust means reliability of the production of software, but In the context of security ,trust means a property of information system which can guarantee it can not be subverted by an adversary without showing any signs to managers of the system.

The measure of trust, to demonstrate a program has an expected property, is another issue, rather than to build a program has the property. Although the producer of software can declare the trustworthiness, but if
there is no rigorous trust measure of status on which the manager of the system could take action, the software could not be “trusted”, statically or dynamically. This is why trusted computing needs rigorous treatment on its concepts to give out a sound measure theory of trust.

There are several ways of modeling the measurement such as probability statistics, software behavior[4], and measure the integrity of static program as data integrity[3], etc. But these methods actually mix different contexts of trust into one, or base their theory on some presumptions of the situations. For example, the measurement of the integrity of static program could not reveal the dynamic semantics of the program. In this paper, we start the topic with analysis and classification of concepts of trusted computing, both for clarity and motivation of our new view of trusted computing. After that, we give a mathematical definition of trust pair which grasps the concept of trust and its measurement, and find close connections between trust pair and one way functions. At last, we reveal that trusted computing has a subtle but deep relation with software watermarking, can be found on the concept of trusted computing.

2. Analysis and classification on the concepts of trusted computing

2.1 The contexts of trust

The requirement of trust varies depends on the contexts, there are roughly three contexts in Fig.1, the production process of software, from the requirement specification to the implementation, that is the process of SE, in this phrase, the purpose is to make the stakeholders to trust the implementation exactly fits the requirements. Next context is the deployment of the object into an environment which is trusted or not, if we force this environment to be “trusted”, for example that we make the system environment totally closed to outsiders and keep out any possible adversaries who might attack on computing units or change their semantics in the environment, the story will end here. But in reality, there is no such guarantee that one system has no security holes which is absolute closed, so conceptually we must build the theory of trust with the presumption of “open system” with possible and dangerous adversaries in it. After deployment, when the units are computing, the measurement of them, statically or dynamically, is reported to a verifier, that may be a manager or a program, to reveal the trust status of the deployed units. We omit the pass of trust in Fig.1, just because the pass of trust could not be viewed as a process of entrust action or origin source of trust.

![Diagram of the contexts of trust](image)

Fig. 1 The Contexts of trust

Actually, there is deference between to reveal whether the units are attacked and to persist original behaviors of them under attack. Like error-correcting-code, detecting an error in a code is easier than correcting it.

So these contexts can be classified into four categories of ‘trust’:

- The trust of quality of software, that trusts the process of production, should not be considered under the name of security, but reliability.
- The trust of the computing environment, building environments to fit security presumptions;
- The measurement of trust, statically or dynamically, the measure can reveal the status of computing units;
The trust of behavior robustness that is persistence of behavior of the computing units when they are attacked by adversaries.

In this paper, the view of trusted computing focuses on the third category, the measurement of trust, because we don’t intend to set any presumption on the ‘open’ environment or to handle the difficulties of behavior persistence, but to build a relationship between the computing units and the measurer of them to make sure the measurement can soundly reveal the status of computing.

2.2 Open environment and adversaries

In cryptograph, communication channels are exposed to adversaries, but for the measurement of trust [5], the environment is open and the computing units are attackable, thus consideration on adversaries is necessary.

The adversary of trust should be considered with only limitation of computation intractability [5] and could have full control of the open environment. Unlike adversaries of crypto, adversary of trust can intercept every input, output and even measurement between computing units and their verifier, and have the full access to the unit itself. Consider the extreme situation called ‘full replay attack’, adversary can replay the information according to the message, the units, especially the verifier send, to deceive the verifier that the units are computing, but in fact they could have been erased from the open environment.

The only way to prevent ‘full replay attack’, in Fig.2, is every time a unit is need to compute some task with certain input, a measure witness should be merged with input, and with the same input, the witness should not be the same, so the measurement could not be the same.

This witness must be constructed to fully relate to every part of the unit code, when the unit is computing; it uses witness as an accessorail input for reflecting its structure. After the witness flows though the computing, it becomes the measurement for one computing process, which can be used by verifier to judge the status. Contrast to the other methods, which measure the unit ‘negatively and passively’, we view trust not a property of the unit, but a close relation between the unit and the verifier, a trust pair. This trust is a bound relation between verifier and unit, ‘trust means this relation could not be deceived by any adversary’.

In the view of computation intractability[5-7], witness can be viewed as, it makes every computing process as solving a intractable problem when giving different witnesses, the unit is the key to the problem, while adversaries could not construct others to replace it, thus the trust of the computing could be soundly measured.

Although adversaries could directly attack the unit itself, such as trying to strip the witness from the unit or to randomly forge some output to deceive the verifier, but this attacks all can be reduced to full replay attack especially in the view of verifier, and since we have no way to model how adversaries could attack on units, we should only assume adversaries are limited by computation intractability. This is why other methods, such as probability statistics and software behavior, will eventually failed, because they ‘passively’ measure the computing unit, but adversaries are subjective, and ‘a trust relation between units and verifier should be constructed rather than passively measured, and more, measure should be found on...
this construct to guarantee its soundness’, this is the most different feature of our definition of trust with the others’, just like the constructing security channels in cryptography.

2.3 The presumption and concept of trust measurement and trust pair

Open environment and trusted environment: In open environment, no a prior trust can be gained. But in trusted environment, we trust any computing unit in it. We can view trust as extending trust relationship from an unit in trusted environment to an unit in open one.

Full replay attack: full replay attack is a unified view on adversaries, they can have all computing history of certain unit to replay and deceive the verifier that the unit is still working. The only limitation is the inherent intractability, although this full access which seems too powerful for the adversaries, but it makes the concepts more precise.

Trust pair and its components: trust pair \((U, V)\) is the pair of unit and verifier. The unit here is not directly the implementation mentioned above but after a construction into a new unit with the verifier, it includes the new unit and a reporter which is just a routing sending everything received from the new unit to the verifier outside the open environment. In conceptual level, it could be omitted. The verifier includes a witness generator, an injector and a checker. Witness generator randomly generates nonrecurring witness in case of replay attack; Injector merges and ‘couples’ input from users and the witness from the verifier into one ‘injection’, and inject this into the computing unit in the open environment as its input; these three parts of verifier could be abbreviated as triple\((I, G, C)\).

Measure process: a measure process starts with a normal functional task, when a user needs, the injector merges the user input and the witness that’s generated by the witness generator then injects them into the unit as its final input. The unit computes with the given input, at any time in that process verifier could ask reporter to report the required information of computing, verifier checks the information to judge the trust status of computing. If verifier does not ask, reporter will send that after halting for judging the status. Especially, if there is no response of reporter, it can be directly concluded that the unit should not be trusted.

Model of components and notations [5-7]: the standard Turing Machine(TM) and nonuniform circuit are used in modeling the components of the pair. Standard notation are used, ‘\(< x >\)’ indicates the representation of x in some computational model; ‘\(|x|\)’ means the length of object x; \(Pr[X]\) means the probability of random variable X. We call two computable functions semantic equivalence if they have the same extension that is they output same y on the same input x. \(p(n)\) denotes a polynomial of n; PPT is the abbreviation of probabilistic polynomial time Turing machine.

3. The definition of trust pair and relation with one way function and software watermarking

3.1 Formal definition of trust pair

Based on the previous analysis, a trust pair is informally defined as an ordered pair \((U, V)\), the verifier is a triple \((I, G, C)\). This pair should be constructed from the original unit \(F\) before deployment to make sure the unit \(U\) is semantically equal to \(F\).

But there is a little difference in the new unit \(U\), because it is constructed to take the inject as input, we should take the injector as a outsider part of \(U\), so the \(F\) is semantically equal to \(<U, I>\), the output and effect of unit is in the open environment, any thing taken out is considered as a part of measurement. To get rid of this, we just consider \(U\) as the intended software, not the original \(F\).

Another issue is the lifetime of unit because of the limited number of witnesses, especially when the input and injection length are limited. When the witnesses are used up, a new pair must be constructed. This limit makes some requirements on \(F\), especially the TM model.

At first, \(F\) should halt in polynomial time of length of input, as to \(U\) the length of injection, So the configuration should of length \(p(n)\). \(F\) has a limitation length of input \(N\), the function of \(F\) is restricted to \(F[n<N]\).

Although this limitation seems wield, but when considering the model of Circuit, it will be natural. And before define trust pair; we define a more loose definition to avoid the issue of semantics.
Definition 1. p trust pair: a pair \((U, V)\) is called p trust pair if in a measure process, for any adversary to attack on \(U\), using any algorithm \(A\), with input of a computing configuration of \(U\), which has the length of \(p(n)\), in the lifetime of \(U\), for any polynomial \(p\), for every input \(x\):

\[\exists w \forall A \in \text{PPT}, \Pr[P(A(U, i_n), x_n, w, U) \neq P(U(i_n), x_n, w, U)] < \frac{1}{p(n)}\]

\(x_n\) is the input by the user, which is a random variable uniformly distributed over \(\{0,1\}^n\); \(w\) is the witness; \(i_n\) is the injection merged with \(x_n\) and \(w\); \(P\) is the predicate which checker use to judge the status of trust; \(U(i_n)\) is the measure of \(U\) computing \(i_n\) in the trusted environment; \(A(U, i_n)\) is the reported measurement of \(U\) computing \(i_n\) in the open environment, under attack of adversaries; above all, all components of trust pair is efficient computable by a deterministic polynomial time algorithm.

The definition actually means \(V\) can trust the property \(P\) in the computing of \(U\) in the open environment under attack by adversaries and actually the pair can be represented as \((U, P)\), since the other components are implicitly related to \(U\) and \(P\) in conceptual level.

In the definition, \(U\) is considered as TM, so there is a restriction on \(F\) and \(U\). But if consider it as nonuniform Circuit \(C\) which is a polynomial size circuit of input \(n\), the restriction is straight forward. In the following, \(U\) is view as a polynomial size circuit.

Trust pair is defined upon p-trust pair, the main issue of trust pair is how to judge the semantics is unchanged according to the reported information \(A(U, i_n)\). It is hard to model it in Turing Machine, since that two Turing Machines are semantically equal is undecidable, while the circuits' semantics is easier to grasp.

A polynomial size circuit[7] with input size \(n\), has two parts: \(p(n)\) logic gates of \(\{\neg, \lor, \land\}\), and the connection graph of them which is a acyclic graph or lattice. If two circuits share the same structure and every logic gate is the same at the same place of the structure, they are called semantically equal. Like the error-correcting-code, we 'embed' the circuit into a structure called ‘coded-circuit’ which encodes the logic gates and the structure of the original circuit. An informal definition of ‘coded-circuit’ is as following.

Definition 2 Coded-circuit: a coded-circuit is constructed on the original circuit, but codes all the components \(\{0, 1, \neg, \land, \lor\}\) respectively and preserves the connection of components. The input of \(\{0 1\}\) for original circuit is coded be the input of coded-circuit; the logic gates of \(\{\neg, \land, \lor\}\) are expanded to more sophisticated circuits or functions of bounded length input to simulate the original logic gates so the original circuit function can be simulated and ‘coded’. We call this simulation as embedding of a circuit into a coded circuit. It should be noted that circuits and coded-circuits could be transformed to TM, such as straight line program. The detail of coded-circuit will be discussed in our further papers.

Not we can found our definition of trust pair on Code-circuit:

Definition 3 trust pair (of coded-circuit): a trust pair of coded-circuit is a pair similar to the p-trust pair, except two different points:

1. \(F\) is a polynomial circuit, \(U\) is a polynomial size coded-circuit; The predicate \(P\) is not a normal predicate, but a combination of two predicate:
   1. \(S^c(c')\): The semantics predicate judges whether the coded-circuit \(c'\) is embedded with a semantically equal ‘normal’ original circuit \(c\).
   2. \(T^c(c')\): The trust predicate judges whether the coded-circuit \(c'\) in the open environment, is simulating the same original circuit \(c\); This predicate actually bases on \(S^c(c')\), since all semantics are related to \(S^c(c')\).

We could view \(P^c(c') = S^c(c') \land T^c(c')\) and trust pair as a special class of p trust pair \((U, P^c)\) and we go on to the subtle relation between p trust pair and one way function.

Theorem 1: a one way function induces a p trust pair

Proof: we give an informal proof here since it is trivial. A (strong) one way function \(f\) is defined as following [5, 7], and we only consider strong one way in this paper:

\[\Pr[A(f(x_n) \neq f^{-1}(f(x_n))] < \frac{1}{p(n)}\]
We construct the function $g$ randomly guessing the reverse of $y$, $g_f(y) = x'$, $f(x) = y$ (actually we can give a constant as output of $g$). If $g$ is put into the open environment, the witness and injection are the same $y$, the property $f(x') \not\in f^{-1}(y)$ could be trust. If not, there must be an adversary who uses a probabilistic polynomial time algorithm $A$ makes the property $f(x') \in f^{-1}(y)$ holds, which contradicts that $f$ is one way.

Since every one way function induces a p trust pair, some certain p trust pair may be transformed into one way function.

**Theorem 2:** Some special p trust pairs induce one way functions.

**Proof:** This proof actually is reverse of theorem 1. $U$ is a function $g$ which always gives constant $c$ (which the user will not input), $f$ is a length preserving and one-one function, $x$ is the input of user, $w$, the witness is just $f(x)$, and injection is $<f, f(x)>$, the predicate is $g_f(f(x)) \neq x$. If the pair $(g, g_f(f(x)) \neq x)$ is a p trust one, by definition:

$$\exists w \in A \in PPT. Pr[P(A(g, f(x), f, x, w, U) \neq P(g(f(x)), f, x, w, U)] < \frac{1}{p(n)}$$

$$\Rightarrow \exists w \in A \in PPT. Pr[P(A(g, f(x), f, x, w, U) \neq x] < \frac{1}{p(n)}$$

since $P$ is $g_f(f(x)) \neq x$, and $g$ is constant function

$$\Rightarrow \forall A \in PPT. Pr[P(A(g, f(x), f, x, w) = x] < \frac{1}{p(n)}$$

since $w = f(x), U = g$, which are already accessible to $A$

$$\Rightarrow \forall A \in PPT. Pr[A(g, f(x), f) \neq x] < \frac{1}{p(n)}$$

since $P$ is $g_f(f(x)) \neq x$ and $g$ is not relevant

$$\Rightarrow \forall A \in PPT. Pr[A(f(x))] < \frac{1}{p(n)}$$

since $f$ is one-one and $g$ is not relevant

So $f$ is one way by definition, induced by trusted pair $(g, g_f(f(x)) \neq x)$.

A conclusion is, from the above theorems, that the concept of one way function is should be stronger than the concept of p trust pair, since a one way function indicates p trust pair, but p trust pair does not necessarily indicate a one way function.

Still, the existence of p trust pair based on the assumption of existence of one way function does not directly indicate the existence of trust pair of coded-circuit. More issues arise, when considered practical requirement that is a demand of efficient methods taking a normal circuit as input and outputting a trust pair of coded circuit. So we have:

**Conjecture 1:** The efficient construct conjecture, if one way function exists, there is an efficient method, that is a polynomial time algorithm taking any normal circuit as input, makes a trust pair of coded circuit.

The reverse of the conjecture is not likely to be true, since reverse of the efficient construct does not give adversaries to attack on the witness. If the randomly chosen witness is nonrecurring, by definition of trust pair, no adversaries could deceive the verifier, even with the ability to access the original circuit. So the definition of trust pair does no demand the hardness of reverse the efficient construct and the definition of trust pair and p-trust pair itself does not necessarily indicate one way functions according to Theorem 2. This indicates the concept of one way function may be more powerful than the concept of p trust pair, which makes us believe that conjecture 1 is true.

**3.2 Construct software watermarking on trust pair**

Software watermarking, or software fingerprint, consists of ensuring that only the software producer can prove in court that he has designed the program [8]. The traditional view of software watermarking is embedding a secret in the software that only could be shown by the producer [9, 10], but this concept misses a point, we called, Integrity of watermarking and the intended software.

Software watermarking requires producer to ‘watermark’ the ‘whole’ software, and prove to judge of the count that only he has ‘that private mark’.
1) **Integrity of watermarking and the intended software**: the mark and software is a whole, and it should be hard for others to strip that mark from software. This integrity is one of the prerequisite, or the others could also declare the ownership after the stripping.

2) **Private knowledge of the mark** [9]: the mark should only be known by the producer and hard to reverse from the software copy. Further more, producer should use ‘the knowledge of mark’ to show the effect which convinces the judge without letting others know the original mark.

The similar is both concepts need the property of ‘integrity’. Trust should be ‘integrated’ into every single part of the code, and every step of computing, and watermark shares the same requirement of integrity, that is no adversaries could possibly ‘deceive’ the judge that he produced the software. In Fig. 3, judge randomly chooses a input \( x \) which is given to the producer and others, waiting for the evidence of verifier to prove who is the owner of the software. In order to show fully control the unit, the producer has two sets of witnesses to switch functions of unit; When using the set of \( w_2 \) as witness, the watermark mode of the coded circuit is activated; while using the set of \( w_1 \), the normal functions of coded circuit is able to users. The judge could trust the producer is the designer of software by the evidence that he can do this for enough times.

Actually, in the process, no one including the producer and the others needs to show the witness they use. They only need to offer the injections according to the input.

![Fig. 3 The scenario of software watermarking](image)

The formalism of the scenario gives the following formal definition of software watermarking triple based on trust pair:

**Definition 4 (strong) software watermarking triple based on trust pair**: A software watermarking triple \((U, V, W^c)\) is a triple based on trust pair. \( U \) is a coded-circuit; \( V \) is the verifier of the trust pair but work in a different way as follows, \( W^c \) is a accessory predicate to judge the software is whether in a switched mode, showing that \( U \) is not in normal status, and:

In the lifetime of \( U \), for every input \( x \):

\[
\exists w_1 \forall A \in PPT, \Pr[P_1(U(U(x), x, w_1, U), x, w_1, U)] < \frac{1}{p(n)}
\]

in which \( P_1(c') = S'(c') \land T'(c') \land \neg W'(c') \)

and \( \exists w_2 \forall A \in PPT, \Pr[P_2(U(U(x), x, w_2, U), x, w_2, U)] < \frac{1}{p(n)} \)

in which \( P_2(c') = S'(c') \land T'(c') \land W'(c') \)

This definition claims that the producer can switch the function of coded-circuit with two set of witnesses, while other can not. When using the set of \( w_2 \) as witness, the watermark mode of the coded circuit is activated, guaranteed by the predicate \( W^c(c') \); while using the set of \( w_1 \), the normal functions of coded circuit is able to users. Loosely speaking, the producer can show to the judge that he can make the
production malfunction with \( w_2 \), respectively the injection of \( w_2, i \), to ‘turn off’ the coded-circuit by achieving the property of \( W^c(c') \), while preserving the property of \( S^c(c') \land T^c(c') \).

1) **Integrity of watermark and the intended software:** this property is ensured by the property of trust, since for different set of witness, the properties \( P^c_1 \) and \( P^c_2 \) of \( U \) is hard to change and should be trusted as the inherent properties of \( U \).

2) **Private knowledge of the mark:** this property is ensured by the property \( P^c_2 \), since only injections are exposed but witnesses are concealed:

\[
\exists w_2 \forall A \in PPT, \Pr[P^c_2(A(U, i^*_n), x_n, w_i U) \neq P^c_2(U(i^*_n), x_n, w_i U)] < \frac{1}{p(n)}
\]

\[
\Rightarrow \exists w_2 \Pr[P^c_2(A(U, i^*_n), x_n, w_i U) \neq P^c_2(U(i^*_n), x_n, w_i U)] \geq \frac{1}{p(n)}
\]

\[
\Rightarrow \forall w_2 \Pr[P^c_2(A(U, i^*_n), x_n, w_i U) \neq P^c_2(U(i^*_n), x_n, w_i U)] < \frac{1}{p(n)}
\]

So it is hard for others to forge an injection to change property \( W^c(c') \), the knowledge of the set of \( w_2 \) should be considered as the private knowledge. The judge could test enough cases to ensure this.

Software watermarking triple \((U, V, W^c)\) is not directly a trust pair since it has two different set of witness, and two predicates respectively. But it could be transformed to a trust pair considering the switch of function as a part of new function. Suppose the original circuit \( C \) for following constructions, a p trust pair for the triple:

\[
(C', S'(c') \land T'(c') \land (Z_i(i) \rightarrow \neg W^c(c')) \land (Z_i(i) \rightarrow W^c(c'))) \text{ is the injector, } Z_i(i) \text{ is true if and only if } \exists w_i(i, x) = I(w_i, x)
\]

The p trust pair of software watermarking triple \((U, V, W^c)\), claims that the coded-circuit \( C' \) has two sets of witness as its switch of function to fit the two predicates of the triple. But since the additional predicate \((Z_i(i) \rightarrow \neg W^c(c')) \land (Z_i(i) \rightarrow W^c(c'))\) makes the p trust pair more complicated that it should be questioned its existence even under the assumption of existence of one way function. The existence of software watermarking triple \((U, V, W^c)\) may imply the existence of one way function, and this will be considered in future work.

4. Future work

The efficient construct conjecture of trust pair is the major concern and the core of the whole theory. Proof of the conjecture requires a quantitative study of coded-circuit, and it may relate to special property of one way functions [7] and p trust pair could be a concept which is strictly weaker than the concept one way function. Although it is strongly believed the conjecture is true, it is possible that it is wrong which means our concepts is too strong for the concept of trust and needing further amendment to the concepts.

The quantitative study of coded-circuit could also reveal the lower bounds of all constructions and components, such as the lower bound of the efficient construct, the lower bound that coded-circuit put ahead to the original circuit, etc. These are very important for practical purpose of the theory.

In this paper, only category 3 of trust is considered, that is trust of measurement, and its concept actually meets the requirement of intrusion detection [11] which needs to detect attack on the intended system, if we extend the trust relation between a single unit and the verifier to a vast trust relation between the verifier and a system which consists of the many trusted units. And these units could interact with each other in a trusted way. This extension could give a formal foundation of intrusion detection.
For category 4 of trust, the trust of behavior robustness, the analogy to software immunity is obvious, since they both require robustness under attack. But the trust of behavior robustness should be analyzed with a more sophisticated model of systems and the adversaries who attack this system; the notation of full replay attack is too powerful and should be restricted.

Since the concept of software watermarking could be founded on p trust pair, there may be other concepts, such as obfuscation of codes [6], etc. The definability of p trust pair bases on the relativism of security that is security is relative, for some it is easy to attack the system, but the others can not. Security starts with distinguishing these differences, it is not reasonable to say a system is absolutely secure, as to a security property.

5. Conclusions

Although the concept of trust has been used in information security for a long time, there is no rigorous treatment on its concept, we begin with analysis and classification of the trust concepts based on the contexts in which the concepts trust is used, and categorize the concepts into four different ones, trust of quality of software, trust of the computing environment, the measurement of trust, trust of behavior robustness, among these we focus on the third concept of trust, the measure of trust.

Contrast to other approaches of the measure of trust, we have two new view points about the measure of trust: ‘trust is a bound relation between verifier and unit, trust means this relation could not be deceived by any adversary’, and ‘a trust relation between units and verifier should be constructed rather than passively measured, and measure should be found on this construct to guarantee its soundness’.

An analogy with cryptography shows a connection between trusted computing and cryptography at conceptual level. The formal definition of p trust pair is given as the base for the further concepts of trust pair of coded circuit. Trust pair characterizes the trusted measure of computing unit in the open environment and coded-circuit is the intended model to implement it. And p trust pair has a close relationship with one way function, a one way function induces a p trust pair while only special p trust pair. The concept of software watermarking could be found on trust pair after an analysis of the scenario of software watermark, and shows the power of definability of p trust pair.

The concept of trust pair arises from the idea called ‘Relativism of security’, that relativism means security property is relative in some context, could not be treated as a property of a single program or system, and should be related to a verifier or a judge. The security of system is the security of the relation between the intended system and the verifier just like the relation between the system and the users of the system. Many concepts of security share this ‘relative’ property and could be given a rigorous formal treatise based on the concept of trust pair.

Acknowledgment

This work has supported by “the Fundamental Research Funds for the Central Universities” from Wuhan University by Grant #20102110101000109.

References


