I. J. Computer Network and Information Security, **2015, 10, 23-34**
Published Online September 2015 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijcnis.2015.10.03

Can universally composable cryptographic protocols be practical?

István Vajda
Technical University of Budapest, Department of Informatics, Budapest, Hungary
Email: vajda@hit.bme.hu

Abstract—The Universal Composability (UC) framework provides provable security guaranties for harsh application environment, where we want to construct protocols which keep security guarantees even when they are concurrently composed with arbitrary number of arbitrary (even hostile) protocols. This is a very strong guarantee. The UC-framework inherently supports the modular design, which allows secure composition of arbitrary number of UC-secure components with an arbitrary protocol. In contrast, traditional analysis and design is a stand alone analysis where security of a single instance is considered, i.e. an instance which is not in potential interaction with any concurrent instances. Furthermore, a typical traditional analysis is informal, i.e. without a formal proof. In spite of these facts, beyond the task of key-exchange this technology have not really took the attention of the community of applied cryptography. From practitioner's point of view the UC-world may seem more or less an academic interest of theoretical cryptographers.

Accordingly we take a pragmatic approach, where we concentrate on meaningful compromises between the assumed adversarial strength, ideality wishes and realization complexity while keeping provable security guarantees within the UC-framework. We believe that even modest but provable goals (especially, if tunable to application scenarios) are interesting if a wider penetration of the UC-technology is desired into the daily-practice of protocol applications.

Index Terms—Cryptographic protocols, provable security, universal composability.

I. Introduction

Modeling of the execution of crypto-protocols in the Universally Composable (UC) framework is far more realistic than in the stand alone classic setting [7], [8]. Familiarity with the UC-technology is at low level in the community of applied cryptography. To master the corresponding techniques (in particular simulation proofs) requires serious investment from a researcher. This fact is worsened by anticipating that the resulted provably secure protocols are not competitive at the practical "market" of efficient protocols. Indeed, the available UC-secure constructions are typically built from inefficient, high complexity elements which typically exclude their practical usage. From this respect the complexity cost of primitives secure even in adaptive adversarial setting is especially painful.

Does it mean that the UC-technology cannot provide tools for daily protocol designers yet? The main message of this paper is a strong belief that this is not the case. There is room for finding practically useful compromises between the assumed adversarial strength, the "wished" strength of the ideal functionality, the assumptions on setups and application scenarios on one side and the computational complexity on the other side while keeping provable security guarantees within the UC-framework.

Obviously, any secure composition from secure elements assumes that the task can be efficiently decomposed to smaller subtasks. Less complex tasks are easier to analyze, in particular the length of a protocol strongly affects the tractability of the corresponding simulation proofs.

At the highest level we consider decomposition along lower and higher layer sub-tasks. In the lower layer we accomplish modularization along the security of communication channels by transforming the protocol into a hybrid in ideal channel functionalities. However, we have to be prepared for adversarial actions even in case when the network adversary is neutralized. This is the case of a higher layer sub-task carried out between mutually distrustful parties: a typical case is when one of the parties intentionally (cautiously) delays the transmission of some information within the run and sends only commitment in advance.

Subsequently we will consider modularization in the lower layer, modularization in the higher layer as well as modularization and abstraction along primitives.

We will use a kind of "macro-substitutes" for computational complexity, as it is closely related to the assumed adversarial strength. On this scale the strongest is the Byzantine adaptive corruption adversary and the mildest is the classic passive man-in-the middle (MIM) network adversary. The strength of the adversary determines the required simulation properties of components (primitives and macro primitives) used in the realization, which are the properties of indistinguishability, extractability and equivocability. These simulation properties provide an advantage for the ideal system adversary (simulator) to be able to simulate the view of the black box real adversary within the ideal system in a way indistinguishable from the one experienced by it in the real system. As we consider the general approach and not a particular construction, we

grab the computational complexity at this high level of simulation requirements and not by the number of public key operations. Similarly we will not touch the communication complexity of construct, as it is typically less costly than high computational complexity in usual scenarios as well as strongly related to concrete implementations.

We distinguish standard, standard secure and UC-secure primitives. Standard primitives are the primitives of daily use, which are not provably secure (e.g. AES symmetric key encryption, RSA public key encryption, standard RSA digital signature), standard secure primitives provably satisfy standard security definitions (e.g. ind cpa/cca2 secure encryption, EU-CMA secure digital signature) while UC-secure primitives (and macro primitives) typically provide additional extractability and/or equivocability properties depending on the adversarial setting. Standard secure primitives support indistinguishability, however the other two properties, especially, equivocability requires enhancement of primitives or macro primitives, which step boosts their computational complexity. The hardest adversarial setting is Byzantine adaptive. If we can avoid adaptivity or circumvent its simulation implications, we can make a considerable step in relaxing the complexity of the design.

There are only a few protocols, which have been found to be UC-secure (with standard secure primitives) and are actually deployed in the real world, coming almost exclusively from the field of key exchange protocols [4], [13], [14], [17], [23], [26]. Here arises the question of what is the practical value of such result for applications if in practice these protocols are used with standard (i.e. not provably secure) primitives.

The computational complexity of a macro primitive indicates the total amount of work performed by participants and is often given in form of costly public-key operations (e.g. modular exponentiations). There is a growing intention among the researchers in the field of UC-security to pay more and more attention to the efficiency problem of the designs of UC-secure macro-primitives (e.g. see [19], [20] for oblivious transfer, [6] for zero knowledge and [27] for commitment task). Still, we think, it is not the best way for a traditional designer aiming to get a view of practical potentials of the UC-technology to dive right away into these special, highly technical works written, as normal, for communication mainly between (top) experts of the field. Instead, we think that highlighting practical trade offs (tuning points) between the strength of formal guaranties, application scenarios and the complexity of realization is a better way for raising an interest. We follow such an approach.

Accordingly, we tried to find a balance between technical aspects and wider accessibility from potential readers. The style of presentation is via showing plenty of examples for applications and proof techniques and summarizing our messages into guidelines accompanied with arguments. We are not aware of other work with a similar goal of making steps of building interface towards traditional (non-UC) designers who are uncertain (or concerned) about the cost of UC-constructs. By space constraint we could not avoid assuming the knowledge some of basic notions of provable security e.g. computational indistinguishability or definition of security by secure emulation of ideal functionality [7], [8].

The structure of the paper is the following. In Section 2 we summarize how the strength of the assumed adversary determines the simulation requirements against the primitives, which in turn affects the complexity of the realization. In Section 3 practical ways of modularization are considered along communication hybrids in the lower layer, macro-primitives in the higher layer as well as modularization and abstraction along primitives. In Section 4 we consider ways to circumvent the difficulties caused by an adaptive adversary, which are the weakening of the ideal functionality and allowing erasures of local data. If we can step back (force back) to semi-honest static setting instead of the high end Byzantine adaptive we can use standard secure primitives without the need for additional extractability and equivocability enhancement.

II. The Adversary

A. Adversarial Model

A standard assumption within the UC-framework is that the network adversary runs an arbitrary efficient algorithm and has access to all communication links as a *man-in-the-middle* (MIM) adversary.

The adversary may also corrupt parties, where the adversary is allowed to choose the parties for corruption. The way of control over a corrupted party can be passive (honest-but-curious or semi-honest) or active (malicious). In *passive* mode the corrupted party continues running the protocol. The goal of the adversary is not to alter the output of parties but to have access to their inputs and longtime secret keys. The adversary tries to avoid detection of the attack for the sake of longtime presence.

In *active* mode there are three things we cannot hope to avoid that parties refuse to participate, substitute their local input or abort the protocol prematurely. In addition to these actions the strongest malicious adversary (called *Byzantine*) can completely deviate from the rules of the protocol, e.g. may arbitrarily modify the function underlying the protocol message. Note, however that such stronger attack is viable only if the honest receiver is not able to verify such deviation from the predetermined rules, which is not the usual case, therefore usually a malicious action means "just" input modification. However, there are protocols where it is difficult for the receiving honest party to detect a Byzantine deviation, e.g. when signatures are exchanged between distrustful parties portion-by-potion.

According to the timing the corruption an adversary can be static (non-adaptive) and adaptive. A *static* adversary controls an arbitrary but fixed set of corrupted parties, fixed before the targeted instance starts running. An *adaptive* adversary is allowed to choose the parties to be corrupted any time during the course of the computation based on the information gathered so far.

Example 2.1: Even, when corruption is carried out by remote hacking into the machine of one of the parties, the hacker will probably stay there for a longer time period which may span several instances consecutive in time. When the adversary resides there it has no reason not to start corruption right from the onset of the instance, even if at the beginning, say, it runs just in the milder passive mode and switches to malicious mode only later time. Another scenario, when adaptive mode is not realistic, is when the time length of the run of the protocol instance is much shorter then the time which elapses from the decision for corruption till the full establishment of the wanted control.

A practical reason behind adaptive operation may be that breaking into a party involves risk/cost, therefore the adversary tries to optimize the timing of corruption and the set of most ”promising” parties to reach the most gain from the action. For example, a dynamically (per instance) selected subset of parties accumulates/stores more sensitive information during the run than the rest of parties and the adversary gets a guess on this subset by processing the information available during the run.

For a concrete example, we can consider the task of secure data collection from a set of sensors via a tree-like communication graph, where the leaves of the tree are the sources of data and the higher order nodes are the collectors. Such a communication graph may be built dynamically up during the run or it may be the case that the roles (leaves, collectors) are assigned a priori. Obviously the collector nodes especially if they are at higher level within the graph are the best targets for breaking into.

The strongest adversary is Byzantine and adaptive. Cryptographic protection against such a powerful adversary requires high complexity enhanced primitives. One of the key issues of this paper is to highlight potential ways to escape or ease this computational burden.

B. Simulation Requirements Versus The Adversarial Strength

First we recall the definition of UC-secure realization: A protocol π is UC-secure realization of ideal functionality φ if for any adversary A there exists an adversary S such that, for any environment Z and on any input, the probability that Z outputs 1 after interacting with A and parties running π differs by at most a negligible amount from the probability that Z outputs 1 after interacting with S and φ [7], [8].

The strength of real adversary A affects the complexity of primitives and protocols via the simulation requirements. This means that the hardness of the task for the simulator boils down to simulation requirements, which are indistinguishability, extractability and equivocability. Simulator S interacts with the ideal functionality and the environment. A black box simulator invokes a copy of the real adversary and runs a simulated interaction of the real adversary. Indistinguishability and equivocability serve to make the view indistinguishable for the simulated real adversary.

We show a summary of relationship between the simulation requirements and the adversarial models in Table 1 and 2, where Table 1 refers to simulation commitments (implicit commitment) while Table 2 shows the case of bit commitment (explicit commitment). We provide no reference to these summary tables as we are not aware of such. It is natural in some sense, as research papers concentrate on concrete construction in given setting (e.g. Byzantine adaptive) and discuss only the corresponding simulation requirements.

Table 1. Simulatability requirements

	Non-corruption network adversary	*Static corruption*	*Adaptive corruption*
Passive mode	indistinguish-ability	indistinguish-ability	equivocability
Active mode		extraction	extraction, equivocability

Table 2. Simulatability requirements in case of explicit commitment (e.g. bit commitment)

	Non-corruption network adversary	*Static corruption*	*Adaptive corruption*
Passive mode	extraction equivocability	equivocability	extraction equivocability
Active mode		extraction equivocability	extraction, equivocability

Guideline 2.1: As simulation requirements directly affect the (computational) complexity of constructs, Table 1 and 2 are very instructive, they provide the first aid in assessing the complexity consequences of different practical scenarios. For instance:

In case of explicit commitment we need equivocability property in all adversarial settings, while in case of implicit commitment it is required only for adaptive adversary. Extraction is always needed for an active corruption adversary.

Looking at Table 2 it may seem strange that passive static corruption setting requires milder requirements then a pure, non-corruption network adversary. Note, however in case of passive static corruption the adversary is aware of the input variable (committed value) from the start of the instance and leaves it unchanged, therefore there is no need for extraction at the simulator. Further applications of the above tables will be shown subsequently.

B1. Indistinguishability

Indistinguishability means that within the ideal system simulator S has to simulate a real message for simulated real adversary A from an (abstract) message received from ideal functionality F such a way that in the view of A the simulated message is computationally indistinguishable from the corresponding message of the real system.

Difficulty of such simulation arises from the fact that at the time of the simulation of a protocol message the simulator is typically unaware of the value of a variable underlying the real message. If this value remains unrevealed then “any” computationally indistinguishable simulated message is sufficient and the simulator will not face a simulation commitment problem.

Looking at Table 1 in passive static adversarial setting this is the only simulation requirements, i.e. this is the mildest case. This implies that all provably secure cryptographic primitives must support indistinguishability requirement (for better transparency we have not included this requirement to each cell of the above tables). The prominent example is the semantically secure encryption under chosen plaintext attack (equivalently, ind-cpa secure encryption).

Note, if the protocol relies on explicit commitment functionality (in a higher layer task), then even for the mildest corruption adversary (passive static) we need primitives (macro primitives) enhanced with equivocability property.

Guideline 2.2: In passive static adversarial setting indistinguishability is the only simulation requirement. Realization with standard secure primitives which by definition support indistinguishability is sufficient (e.g. ind-cpa secure encryption).

Indistinguishability is a generic property which we require in the simulation in all adversarial settings as it is inherently associated to the underlying paradigm of secure emulation.

B2. Extractability

Assume a malicious (active) adversary *A* corrupts a party and falsifies a non-local variable. Extractability means the requirement that simulator *S* has to be able to extract the changed value of this variable from the real message output by (black box) adversary *A* as the simulator needs this value to form an input message for ideal functionality *F*.

If corruption is carried out by a weaker, passive adversary and we consider implicit (simulation) commitment then there is no need for such extraction capability by simulator *S*. Indeed, by definition the simulator has access to the inner state of the corrupted party within the ideal system therefore *S* sees also all non-local variables which will not be modified by a passive adversary *A*.

In lower layer tasks implementation of extractability is often straightforward when this property is inherently supported by standard secure primitives without additional enhancement (trapdoor functions). In case of (public key) encryption the corresponding decryption provides the extraction of bits of the plaintexts. In case of digital signature the signed content is sent also in plain, i.e. there is no need for extraction. Consequently, it is no wonder, that the UC-analysis of well known lower layer protocols is almost straightforward concerning the simulation proof. In this respect, for example, we refer to the analysis of TLS protocol under malicious static adversary [17]. In the higher layer of protocols, in contrast, macro primitives (e.g. commitment, zero knowledge) require an "UC-upgrade" in order to build extraction capability into them.

Guideline 2.3: Lower layer protocols built from ind-cca2 trapdoor primitives and EU-CMA-secure (Existential Unforgeability under Chosen Message Attack) digital signatures can provide UC-security assuming non-adaptive adversary as the extraction capability is inherently supported.

B3. Equivocability

Equivocability property is needed when simulator *S* has to commit to a value which is unknown for it at the time of (implicit or explicit) commitment, however becomes revealed later during the run of the instance. Equivocability allows the opening of the commitment consistently to any value which turns up later as the true value.

Recall, in case of explicit commitment opening is part of the protocol itself, while in case of implicit (or simulation) commitment the true value becomes known later via adaptive corruption.

Example 2.2: In case of symmetric key encryption, one time pad is the only known encryption scheme which provides equivocability: the simulated ciphertext is chosen randomly from the space of ciphertexts, and later on the simulated key can easily be adapted to any (true) message. Reference [13] shows a classic application of changing from symmetric block encryption to computational one time pad in order to assure equivocability property in symmetric key setting: secure PRG is used with timely erasure of the secret seed where the seed is obtained from a previous secure key exchange.

In case of semantically secure public key encryption and static setting the simulator chooses an a priori fixed message. In adaptive case substantial (highly complex) enhancement of standard secure primitive, so called non-committing encryption is needed to provide equivocability.

Guideline 2.4: Whenever possible it is advisable to escape the need for the implementation of the equivocability property:

i) There are practically interesting application scenarios where the adaptive model is non-realistic (see Example 2.1). In such scenarios we can avoid the equivocability requirement by assuming that the corresponding realization does not use bit commitment modules.
ii) There exist techniques, which allow that static secure protocols become secure even against an adaptive adversary. Such techniques are weakening of the ideal functionality and erasure of certain local data (see also in Chapter 4).

III. Modularization

The strong (universal) composition operation is in the heart of the UC-framework. Trivially, if we really want to exploit this capability, we have to be able to decompose. Modularization can be done along different coordinates as

i) subtasks, the original cryptographic task itself is broken into;

ii) primitives, macro primitives, modules, setups, trusted parties used in the realization of the ideal functionality;
iii) the set of guaranties provided by the ideal functionality.

The most straightforward example in set i) is a task comprising of a series of subtasks carried out one after another in time (e.g. initialization (setup) → party authentication → key exchange → secure message transmission → secure communication session). The prominent technique in this set of modularizations is the separation of communication security from the higher layer component.

From set ii) we mention protocols where modularization is done in the higher layer by applying macro primitives (bit commitment, zero knowledge, oblivious transfer) or special primitives that are used to directly realize some secure function evaluation subtask (e.g. homomorphic encryption for accumulation of encrypted inputs into an encrypted output).

Standard security guaranties provided by any ideal functionality are secrecy, correctness and robustness. Here standard means that it follows from the intuitive notion of an ideal functionality within the UC-framework. Indeed, correctness and secrecy are automatically satisfied since an input to the ideal functionality is not revealed in any way to the adversary and the ideal functionality does compute the result correctly. Robustness follows too; in the UC framework, we can corrupt parties and still have a good simulation in the ideal system (up to the level, tolerated by the ideal functionality). Beyond these standard guaranties an ideal functionality may provide various further guaranties, e.g. fairness, anonymity, deniability.

A. Modularization In The Lower Layer

Separation of the security of the communication channels (lower layer) from the higher layer (sub-)task is a standard approach in the UC-framework. The main lower layer tasks are party and message authentication, secret message sending and key exchange.

Here we demonstrate this modularization step via examples. We will refer to the following type of communication channels: raw, authenticated (auth), message authentication (mauth), secure (sec) and confidential (conf) message transmission channels. Ideal functionalities for authenticated (F_{AUTH}) and secure message transmission (F_{SMT}) are defined in [8]. Ideal functionalities for raw and confidential channels, as examples are given below.

Example 3.1: Intuitively, raw transmission guaranties no security at all:

Upon receiving input $(\text{sent}, sid, \text{m})$ from party A, do: if $sid = (\text{A}, \text{B}, sid')$ for some message m then send $(\text{sent}, sid, \text{m})$ to the adversary, then after the adversary replies with a message $(\text{sid}'', \text{m}')$, output $(\text{sent}, sid'', \text{m}')$ to party B, where $sid'' = (\text{X}, \text{B}, sid')$ for arbitrary party X and message m'. Else ignore the input.

Example 3.2: Recall a secret channel (secure message transmission) provides privacy and authenticity services, where authenticated transmission means integrity protection as well as party authentication. By *confidential transmission* of a message m we mean a weakened secret channel which does not provide full authentication just integrity protection. For intuition, consider the example when a formatted message arrives to a party encrypted by public key encryption, i.e. sent by "somebody". The corresponding ideal functionality is the appropriate modification of F_{SMT} [8], where the modified part is the following:

Output $(\text{sent}, sid, \text{m})$ is sent to party B, output $(\text{send}, sid, l(\text{m}))$ is sent to the adversary, where $l(\text{m})$ is the leakage function, furthermore $sid = (\text{B}, sid')$ for some unique sid' (unique for messages to B)….

Note, party B is not informed about the identity of the sender (via the sid).

Subsequently we show examples for modularization and idealization along communication channels. We will do it on short, classic protocols as their length allows considering several of them and allows demonstrating different aspects of the approach within a few steps.

Though the general approach of designing higher layer protocols over ideally secure or ideally authenticated channels is standard within the UC-framework, the method we show below is specific. The general approach suits the case when we design a new UC-secure protocol and the same type of ideal channel is assumed under each (higher layer) protocol message (e.g. an F_{AUTH}-hybrid protocol is designed). In contrast, here our aim is to analyze existing protocols which originally were not designed within the UC-framework. We will fit the corresponding ideal channel to each individual message within the protocol, or even a message is broken into parts and different parts are transmitted over different type of channels. Here we assume that we are able to identify those channels. Identification means that we can determine the type of the channel (e.g. authentic, confidential, secret etc.) as well as we can parse the corresponding cryptographic elements, their arguments and parameters. Reasoning behind such assumption is that all potential techniques for cryptographic realization of these sub-tasks are known. Furthermore, parsing of messages should have to be made correctly by the description of the protocol. Separation and idealization of channels help to get a clearer and simpler view of the protocol as the channel-hybrid will only contain cryptographic operations which implement the upper layer task furthermore the UC-secure implementation of the channels can be analyzed independently.

Example 3.3:

Preprocessing

The analysis starts with preprocessing. If a message of the protocol can be parsed to elements corresponding to different type of channels we send those elements separately in consecutive protocol steps. The aim is to use appropriate channel for each message or message part. There are protocols where parties are used also for

forwarding (relaying) messages. We substitute such relaying by direct sending. We will use the following notation in the hybrid protocol:

$$B \rightarrow C: M \,/\, \text{channel\_type (attributes)}$$

where information *M* is sent from party B to party C over a channel of type *channel_type* with optional channel *attributes* (e.g. session id, leakage function).

For a concrete instance, hybridization of the modified Otway-Rees session key transfer protocol is shown below (corresponding rows of the hybrid are pointed by →):

Setup: secret encryption keys K_{AS} and K_{BS}

1. A→B: A, (B), N_A → -/raw ($sid_1 = (A,B,sid_1')$)
2. B→S: A, B, N_A, N_B →-/raw ($sid_2 = (A,B,sid_2')$)
3. S→B: $\{N_B, A, B, k\}K_{BS}$ → k /sec ($sid_3 = (A,B,sid_3')$)
4. S→A: $\{N_A, A, B, k\}K_{AS}$ →k /sec ($sid_4 = (A,B,sid_4')$)

The setup of channel parameters (e.g. secret keys) is prepared by initial setups and during steps preceding the use of the actual channel. Accordingly, we analyze the security of the realization of channels one-by-one, following their order within the instance. Security of a channel within an instance assumes the security of all preceding channels. The steps of the analysis are the following:

i) idealization of channels per message base,
ii) analysis of the UC-security of the hybrid,
iii) analysis of the UC-secure realization of channels.

When the task is of lower layer then step ii) is typically trivial as there remains no further cryptographic operation within the channel-hybrid protocol. In step iii) the essential technical task is to ensure (cryptographic) separation of different channels inside the instance and within different instances. During an analysis we may also stumble into usual "tricks" in traditional protocol practice such as implicit reference, overloading or state dependence, which naturally clear up in an UC-revision. For example, in the above instance encryption of redundant message wishes to realize both authenticity and privacy.

Example protocols

As the protocols below are well known and we refer to them in their original form, for the sake of space saving, we will not explain the notations of protocol elements in details. We will assume static adversary.

Protocol 1 (public key Needham-Schroeder key exchange protocol)*:*

Setup: public keys PK_A and PK_B

1. A→ B : E_{PKB} (N_A, A)
   → A, N_A /conf ($sid_1 = (B, sid_1' = N_A)$)
2. B→ A : E_{PKA}(N_A, N_B, B)
   → B, N_A, N_B /conf ($sid_2 = (A, sid_2' = (N_A, N_B))$
3. A→ B : E_{PKB} (N_B)
   → N_B /conf ($sid_3 = (B, sid_3' = N_B)$)
4. A, B → Z: $k = N_A + N_B \pmod 2$

Note, random nonce N_A and N_B serve not only as ingredients for the common session key k but implicitly they are used also as part of *sid*. This overload is formalized in the hybrid by duplicating the corresponding elements.

First we consider the security of the hybrid protocol. We have to simulate the hybrid protocol in the ideal system with ideal key exchange functionality F_{KE} [8]. There are no inputs just public identifiers (A, B) and locally generated random elements (N_A, N_B). First we consider a pure network adversary (MIM adversary), and next a static corruption adversary.

Assume the goal of the network adversary is to agree in a common key with honest party B in the name of party A. The essential steps of the simulation are the following. The simulated (real) adversary assumes sending input message $m_1 = (A, N_A')$ to party B via the first confidential channel however the message is intercepted by the simulator. The simulator sends message (send, sid_2', $l(m_2 = (B, N_A', N_B'))$) to the (simulated) adversary as this is the view of a network adversary of the second confidential channel. The adversary is unable to carry out its attack successfully. When it sends a message $m_3 = x$ the simulator drops the message. Note, in this case the simulator does not interact with the ideal functionality.

Assume now that initiator party A is corrupted (both in the real and the ideal systems). The simulated (real) adversary assumes sending input $m_1 = (A, N_A')$ to party B via the first confidential channel however it arrives to the simulator. The simulator waits until it gets key k output by the ideal functionality F_{KE}. Then the simulator sends message $m_2 = (B, N_A', N_B')$ to the simulated adversary, where $N_B' = k - N_A' \pmod 2$ (in the syntax of the second confidential channel). The simulated adversary sends message $m_3 = N_B$ (as an input to the third channel), which is dropped by the simulator. Simulation in case of corrupted responder (party B) is similar.

In static setting the channels can be realized by semantically secure (ind-cpa) public key encryptions.

Protocol 2 (Kerberos symmetric key exchange protocol)

Setup: secret keys K_{AS} and K_{BS}

1. A→ S: A, B, N_A → - /raw ($sid_1 = (A,B,sid_1')$)
2. S→ A:$\{k, N_A, L, B\}K_{AS}$ → k /sec ($sid_2 = (A,B,sid_2')$)
3. S→ B: $\{k, A, L\}K_{BS}$ → k/sec ($sid_3 = (A,B,sid_3')$)

Intermediate setup: session key k

4. A→ B: $\{A, T_A\}k$ → A,T_A /mauth ($sid_4 = (A,B,sid_4')$)

5. B→ A: $\{T_A\}k \quad \rightarrow T_A$ /mauth $(sid_5 = (A,B,sid_5'))$
6. A, B→ Z: k

In steps 4 and 5 the aim is to provide confirmation of knowledge of common key k. In both directions the same key (k) is used to encrypt a nonce value (T_A) and its extension with identifier A. It is not trivial to find an intuitive ideal analogue to these steps of proof of knowledge relying just on our secure transmission elements (conf, auth and sec). The closest concept is message authentication based on symmetric key encryption. Indeed, as the messages here are not private, we would not degrade the security by prefixing them to the ciphertexts and this way we would obtain messages with MACs (Message Authentication Code). Recall, within the UC-framework we can resolve the problem of dependence by common keys by applying the JUC (Joint state UC) composition operation [16]. Applying JUC for this concrete problem we simply have to ensure that the message in each dependent MAC-channel carries unique identifier. Time parameter T_A assures such uniqueness. (Key k is probably used also in protocol messages which follow this key agreement, where we also have to resolve dependency via JUC.)

Protocol 3 (Chain of digital signatures)

A→ B: $(\text{Sign}^{(n)}(\ldots\text{Sign}^{(2)}(\text{Sign}^{(1)}(m_1), m_2)\ldots)\, m_n)$

For preprocessing, let's decompose this message into n sub-messages each with a single signature:

1) A→ B: $\text{Sign}^{(1)}(m_1)$

2) A→ B: $\text{Sign}^{(2)}(\text{“}\text{Sign}^{(1)}(m_1)\text{”}, m_2)$
…
n) A→ B: $\text{Sign}^{(n)}(\text{“}\text{Sign}^{(n-1)}(\ldots, m_{n-1})\text{”}, m_n)$

where "xxx" is just a bit-string, i.e. it is not considered as representation of a cryptographic operation. For instance, when party B processes message m_2 then it verifies $\text{Sign}^{(2)}$ on message ["$\text{Sign}^{(1)}(m_1)$", m_2]. This way we derive n signature based authenticated channels.

We refer to the signature-based authentication protocol (SBA) and the claim that this protocol UC-securely emulates ideal functionality F_{AUTH} (authenticated message sending) in the hybrid of certification ideal functionality (F_{CERT}) [9]. Accordingly, for UC-secure realization of chained digital signature (and the digital signature, in general) we require public key certificates to be sent along signatures, where the (realized) signature algorithms provide EU-CMA-security guaranties.

An example for the chained signature is the implementation of an authenticated route in a communication graph. The authenticated route is an extension of the authenticated channel (authenticated link), which latter is a one hop route only. For an application we refer to secure route acquisition protocols in sensor network environment [33], [35], [36].

Here we make a small detour to step ii) of the analysis, the analysis of channel-hybrids. Intuitively, after the idealization of real channels the remaining cryptographic operations serve exclusively the protection of honest but distrustful parties. Consequently, if we fully hybridize such a protocol (both in communication channels and higher layer subtasks), the resulting hybrid already contains no cryptographic operations. Intuitively, we have neutralized all adversarial attacks: as if there were no network adversary and as legitimate parties would give up dishonest behavior. One might guess that for full-hybrid protocols UC-security simplifies to guaranteeing (verifying) correctness, i.e. the predetermined input-output functionality of honest parties in the view of the environment. The problem is a bit more complex we show in the next example.

Example 3.4: Consider the following simple 3-party functionality: parties A and B get inputs m_1 and m_2, respectively and party C outputs $m_1 + m_2$.

Natural realization is secure transmission of the corresponding messages from A and B to C. Obviously, the channel-hybrid protocol corresponding to this realization is functionally correct (party C outputs $m_1 + m_2$). Assume now a realization where, additionally, one of the parties A and B shares its input with the other party (over a secure channel). Though this version remains functionally correct, it is not UC-secure. Indeed, in case of corruption the outputs of the adversaries in the real and ideal systems will differ. We generalize this example as follows:

A functionally correct full-hybrid protocol is UC-secure if and only if none of the parties gets to know any input information which is not allowed by the ideal functionality (of the task).

The argument is the following. In case of corruption the simulator also gets access to all inputs, outputs as well as all messages received or generated by corrupted parties up to the time of corruption. Recall, a black box simulator plays the role of an "interpreter" between the (a simulated copy of the real) adversary and the ideal functionality. Accordingly, a simulation problem may happen by the following two reasons: i) the information the simulator gets from the ideal adversary is insufficient to simulate an indistinguishable real message for the adversary or ii) the simulator is not able to extract input variables from a message generated by the real adversary. Parties transmit and receive data elements such as public elements (e.g. party identifiers), locally generated random elements (random keys, random nonce), inputs received from the environment as well as some mappings of such elements. According to our assumption of fully hybridized protocol, case i) of simulation problem will not happen. However, case ii) may happen when in the real system some parties receive input elements from other parties during the run which are not available for the simulator in the ideal system (recall the above example).

The practical benefit of this example is that it simplifies the assessment of the UC-security of the full-hybrid protocol. After we have verified that the hybrid

protocol is functionally correct, we check the state of parties at the end of the run for the presence of input values (in general, input information) which they are not allowed to possess according to the definition of the ideal functionality. In the above key exchange protocols we did not need this observation as there were not input messages.

B. Modularization In The Higher Layer

Higher layer tasks are typically carried out between distrustful parties. Corresponding applications appear, for example, in tasks of e-voting, e-auctions, e-gaming, e-gambling, contract signing or secured database lookup. Designers of these protocols apply modules, overwhelmingly commitment and zero knowledge modules. The reason is that such modules fit to intuitive meaning of the task or deter parties from deviation from the rules of the protocol. For example, commitment is the essence of bidding in e-auction. Another example is non-repudiable commitment in e-gaming and e-gambling protocols, where such modules guarantee that players' misbehavior will be detected.

Classic results on modularization of the higher layer were given in [38] and [21], extended for the UC-framework in [15]. The GMW protocol compiler assumes we have constructed a protocol for the semi-honest model (in stand alone and static setting). The compiler outputs a protocol which is secure even against a malicious adversary: we allow for such an adversary to choose the local input (m') and the random-pad (r') arbitrarily (from the predetermined range), however we want to force it to calculate the protocol message (M) according to the predetermined function f, i.e. M = f(m', r'). This means that input–output behavior of the protocol output by the compiler under any attack from a malicious adversary can be reproduced for the original protocol under some attack from a semi-honest adversary.

Recall, the main steps of forcing the wanted behavior are the following:

At the beginning of the run parties commit to their local input (which input potentially has been maliciously modified) and also to their local random-pad. In both these commitments the committing parties also prove by zero knowledge proof of knowledge that they know corresponding opening information. Intuitively, from this point on parties are securely bound to these inputs and random-pads anyhow those were chosen. Indeed, because of the soundness property of the proof, we know that the user must really act honestly in order to be able to provide a valid proof. Furthermore, because of zero knowledge property, we know that the user does not compromise the privacy of its secrets (input, random-pad). During the run a party actually sending a protocol message proves that the message has been calculated by the algorithm defined by the protocol with inputs (local input, local random-pad) fixed at the beginning of the run.

Subsequent examples show application scenarios, where malicious behavior would harm correctness without efficient countermeasures.

Example 3.5: Consider a task of secure data collection, where the communication network is a tree, with leaves as data sources. Each data element is an integer value from range (0,1,…N). Intermediate nodes forward data toward the root, which calculates some statistics (e.g. average). Assume neither the forwarding nodes nor the root is allowed to see the individual values. A malicious data source could significantly distort the result by selecting a value well beyond N. To prevent such deviation each data source should provide a zero knowledge proof that his value is within the legitimate range.

For another application scenario, we can consider *e-voting* with parameter value $N = 1$, i.e. there are only two candidates [22]. Another scenario is the task of *e-auction* with Vickrey-type of auction, where each bidding party sends its bid encrypted to the broker and the highest bidder pays only the second-highest bid. A malicious bidder may be able to ensure his winning by sending an invalid high bid while paying only a "normal" price [28].

Example 3.6: An "exotic" application scenario is an *electronic card game* between with remote parties without common trusted functionality (e.g. poker over Internet). Parties have to be forced to do the steps of the game correctly like distributed shuffling of deck, dealing of cards, drawing or opening of cards. In this application zero knowledge proofs are used extensively [32]. □

Example 3.7: In the tasks of an *e-contract* distrusting parties cautiously exchange signatures portion by portion. Zero knowledge proof is used to prove that a portion of a signature is prepared correctly (note, the correctness of a portion cannot be verified by standard signature verification). We see once again the case that if the protocol is designed for semi-honest adversary then honest parties might not be able to detect that a party deviated from the rules of the protocol. Therefore we have to upgrade the protocol by adding zero knowledge proof to it [24].

Recall, semi-honest corruption adversary does not alter the input values. In the above examples the adversary was allowed to change values arbitrarily though forced to do it within the predetermined range. It is not a contradiction. Note the latter values are not inputs from the environment (by the model) but correspond to a local variable, e.g. the party decides on his own vote or bid.

Guideline 3.1: In contrast to the malicious adversarial setting, when the adversary can be forced back to semi-honest behavior with appropriate algorithmic measures (in particular, zero knowledge proofs), in general, we cannot compel the adversary to choose static corruption instead of adaptive.

Indeed, if the adversary is passive then we are not able to detect the moment when the adversary starts corrupting a party. Timing of adaptive corruption cannot be influenced by direct algorithmic measures. Instead, the application scenario should prevent adaptive such attacks.

Guideline 3.2: There are two algorithmic countermeasures against malicious behavior. One of them is forcing such adversary to give up malicious behavior and change to be semi-honest. This technique uses

commitment and zero knowledge macro primitives in the higher layer. Handling the "weak" malicious behavior where the adversary substitutes input values but does not deviate from the usage of correct mappings when it generates messages, we rely on extraction capabilities at corresponding primitives. This latter technique of extraction appears both in the lower and the higher layer.

Except [22] constructions referred in examples 3.7-9 did not set the goal of providing the strong UC-secure guaranties. Generic transformations from Σ-protocols to UC-ZK protocols are known [18], unfortunately, they come along with a significant computational overhead, making the resulting protocols impracticable for real-world usage (at least as of yet). The most efficient UC-secure zero-knowledge proofs of knowledge have been proposed in [6] so far.

C. Modularization And Abstraction Along Primitives

From the point of view of the efficiency of analysis within the UC-framework, one of the most promising techniques is the cryptographically sound Dolev-Yao style symbolic analysis [12]. Here cryptographic primitives are substituted by their symbolic models. If this way we could substitute all the primitives within the protocol, we could get a fully symbolic version of the protocol amenable even to automatic analysis by using standard tools. The approach in [12] generates such symbolic version in two steps. The first step is done within the UC framework where the primitives are substituted by the corresponding ideal functionalities. In the second step the resulted semi-abstract protocol is converted into a symbolic protocol. As of yet the available primitives within this approach are public key encryption ([12]) and digital signature [30]. The key theoretical point is that we can substantially simplify the analysis without loosing cryptographic soundness. The simplification of the analysis provides the potential for analyzing larger protocols, assuming that the protocol is built from the mentioned primitives only. Recall, in contrast, carrying out reduction proofs is tractable only for smaller protocols (and this is the main pressure for searching ways to modularization).

Similar approach has been suggested in the works of Backes, Pfitzmann and Waidner (BPW) [1], [2], [3]. This approach falls behind [12] in several respects. It requires from the analyst direct reasoning about protocols within a full-fledged cryptographic model. Consequently, this approach retains much of the original complexity of the problem (asymptotic nature and complexity bounds). Big practical advantage of the approach [12] is that the (potentially) automated security analysis can be done to a single session, and it is guaranteed that the verified security properties will be retained for the overall system, even when this system consists of an unbounded number of sessions, and is not fully known in advance. In contrast, in the BPW's approach the symbolic analysis is applied to the entire system as a whole, i.e. for all sessions and all protocols that run concurrently with the target instance. On the positive side the BPW's approach provides richer library of primitives for symbolic substitution. For practical application of the BPW's approach we refer to [34], [35], [36].

IV. Lowering The Complexity

In the previous section we have seen tools which help to force malicious behavior back to semi-honest. As a result semi-honest designs are upgraded to stand attacks even from a malicious adversary. Now we are interested in tempering the complexity consequences of adaptive corruption and finding ways to upgrade designs to adaptive made originally for static setting.

A. Weakening of The Ideal Functionality

The idea is the elimination of the need for the equivocation property via weakening of the ideal functionality. One such technique includes an appropriate non-information oracle in the ideal functionality. This technique was proposed in [13] and demonstrated for the 2-step DH key exchange as well as for a symmetric key encryption based secure channel protocol. Essentially, such an oracle exempts the simulator from the task of simulation of real protocol message based on message received from the ideal functionality. The oracle provides the simulator with the real message such a way that the simulator gains no additional information it would not have in a simulation without the help of the oracle.

The requirement of providing no information is inherently connected to the concept of semantic security. In the mentioned applications in [13] the DDH assumption as well as semantically secure symmetric key encryption was assumed, respectively. Any semantically secure mapping which is involved in simulation commitment can be used in non-information oracle in order to eliminate the need for equivocability. However the need for extraction capability cannot be eliminated, in general, because in case of a malicious adversary the "weakest" action is changing the inputs of the corrupted party in an arbitrary way. Therefore a goal to solve cryptographic tasks in adaptive setting using "just" standard secure primitives can be reached, in general, only if we assume a semi-honest corruption adversary (together with weakened ideal functionality), as in semi-honest static setting indistinguishability is the only simulation requirement. Exception is when the only primitive involved in simulation commitment problem is a trapdoor permutation which can inherently provide extraction capability without extractability enhancement.

B. Erasure of Local Data

During the run of the protocol trusted erasure of local data not needed any longer may boost the level of security by a natural way.

A model for the erasure technique is the following. Let $z = F(\text{input}, \text{local})$ a value sent in the real protocol, where F is a deterministic mapping. Let $z' = F(\text{input}', \text{local}')$ the corresponding value simulated in the ideal system without knowing value *input*. If value *local* is erased (in the real protocol) after *z* has been

computed, the simulator is exempt from the potential commitment problem of producing a value *local"* such that $z' = F(input, local")$. This means we eliminate the need for equivocability.

In general, trusted erasure may help to eliminate equivocability requirement or at least to reduce the complexity of the realization of such a requirement.

Example 4.1: An intuitive example for the elimination of the commitment problem in adaptive setting is an e-voting application, where we erasure the encrypted individual votes, right after they are securely accumulated by homomorphic encryption [22]. For an example for lowering the complexity of the equivocality property, we can refer to a classic erasure application in case of symmetric key encryption. Here we change from semantically secure block encryption to OTP encryption based on secure PRG with timely erasure of the secret seed of the PRG (where the seed is the output of a previous secure key exchange). Recall, by using PRG we get a computationally secure equivalent of OTP, where equivocability is straightforward [13].□

A proof for "adaptive UC-security without erasure" for a design is surely of higher scientific value than "with erasure". However, if under efficiency considerations the practical usage of the design has no perspective then the scientific value is questionable for a practitioner. The usual objection to erasures concerns its trust-ability, typically by the following two arguments: the information from magnetic storage media cannot be erased perfectly as well as when the protocol runs on general purpose firmware such as Windows machines hanging on the Internet, a party cannot be sure that all backups have also been eliminated. It is a trade-off that weak infrastructure requires heavier algorithmic security components. Our point here is that if we want to prevent the blow up of computational complexity of implementations and to keep the strong guaranties of UC-security at the same time, we must support it by stronger firmware at the parties. For example, this may mean the use of security-purpose (but commercial) integrated circuits or modules where all crypto-related data is localized and the erasure is guaranteed by the design of the firmware element.

Can we always achieve UC-security in adaptive setting if we undertake the complexity consequences? It seems the answer is negative, as it is guessed that in general even with trusted erasure capabilities we cannot attain security in adaptive setting. Concretely, [37] presents a guess that we cannot step higher than static security, as even forward security (weak case of adaptive security when corruption can happen just before or after the run) cannot be attained for longtime authenticators even if we can erase local data. Here we provide a supporting argument to this guess:

Because, secure identification (F_{ID}) can (trivially) be realized over F_{AUTH}-hybrid, it is enough to consider the conjecture for F_{ID}. Consider the following protocol model (KeyGen, I, V) for identification:

We take it plausible, that it is necessary for a party of an identification procedure to have a unique public identifier and a related secret element *k*, generated by algorithm KeyGen.

Assume a two-party identification task, where party B authenticates party C. During this authentication, party C receives (in one or more steps) bit-string $i = I(r_B, r_C, k_B, pub)$, where algorithm I is public; r_B and r_C are temporary local random elements generated by party B and C, respectively; k_B is the unique secret element of party B; pub is public information (e.g. public identifiers, public keys etc.).

Party C runs a public verification algorithm V with input (i, r_C, pub). Assume, after the run of an instance, honest parties erase temporary local data elements, here, random elements r_B (r_C). As long-time authentication is considered therefore the secret elements are not erased. The adversary sees value i.

Assume now that the adversary corrupts party B after the run of the instance of the protocol and gets access to k_B. Forward security cannot be achieved, because, as soon as, the adversary becomes aware of the key, he will be able to distinguish a simulated transcript from a real one. Indeed, there should exist such a distinguishing algorithm, otherwise, anybody could impersonate the party successfully without knowing the secret information.

C. Using Standard Primitives?

Recall, several lower layer protocols which are in practical use (mostly key exchange protocols) have been analyzed in the UC-framework and found secure in specific adversarial settings. What is the implied message of these results for the daily usage of such protocols? Namely, these results on UC-security are valid only when the primitives in those protocols are UC-secure (in the given adversarial setting), i.e. if we substitute the standard primitives by corresponding provably secure ones. The point is that by such substitution the original practical efficiency is typically eliminated. Accordingly, we raise the following "strange" pragmatic question: is it possible to provide any provable guaranties if we keep the original fast non-UC-secure primitives? The composition theorem suggests that we do loose all guaranties. Really all guaranties are lost?

In the subsequent intuition we use the conventional notations from [7], [8]. Assume a primitive ρ UC-emulates ideal functionality φ , furthermore protocol π^{ρ} UC-emulates hybrid π^{φ} . Assume that hybrid π^{φ}, UC-emulates the ideal functionality G of the task. A protocol or primitive is successfully attacked if it can be distinguished from its ideal version with non-negligible probability. By the proof of the UC-theorem ([8]) if a (PPT) algorithm h successfully distinguishes protocols π^{ρ} and π^{φ} then h can be used to successfully (with non-negligible probability) distinguish also ρ and φ (at least one instance of them from those called by π concurrently). Now assume we substitute primitive ρ with a standard primitive ρ' under the following empirical assumption: we take the risk (thought to be negligible) that in any practical application scenario there appears an

efficient algorithm able to break ρ' by the security assumption made for primitive ρ (e.g. an AES ciphertext reveals some partial information about the underlying plaintext, i.e. a "practical" semantic security assumption is broken). In other words, any attacking (distinguishing) algorithm against ρ' is assumed to come only from a subset H of all PPT algorithms (scaled in complexity by security parameter). Via the mentioned proof of the UC theorem set H implies a set H' of algorithms aiming to break (distinguish) protocols π^{ρ} and π^{φ}. It follows that protocols π^{ρ} and π^{φ} also cannot be distinguished successfully (conditioned on attack set H'). As π^{φ} UC-emulates G (they are indistinguishable for any PPT algorithm), we might conclude that π^{ρ} and G are also indistinguishable in the restricted attack environment. We could say that π^{ρ} "conditionally UC-emulates" G.

Note, intuitively, here we would like to separate the provable security of the "UC-logic" of the protocol (i.e. π^{φ} UC-emulates G) from the assumption about the UC-security of a primitive. For example, the correct "UC-logic" is responsible for secure separation of the target instance in the environment of concurrently running arbitrary protocols. As a closing word, we reiterate that here we raised the question of what security guarantee remains if any if we substitute an UC-primitive with conditionally secure one.

V. Conclusions

Universal composability provides the available strongest guarantees for the security of cryptographic protocols. The price for provable guaranties is paid in terms of computationial complexity. One way to lessen this handicap in efficiency is via designing more efficient UC-secure primitives and macro primitives. The other and more pragmatic way is to look for practical trade-offs between the strength of formal guaranties, application scenarios and the complexity of realization. We followed and promoted the latter way in this paper. Our main focus was on the adaptive Byzantine corruption mode of the adversary because this mode forces the highest increase in complexity. The root of this increase is manifested in the simulation requirements of extractability and especially in equivocability. We highlighted several techniques to ease the complexity burden, which are weakening of the ideal functionality, allowing erasures of local data as well as stepping back (forcing back) to semi-honest static setting. We strongly believe that even modest but (formally) provable goals, especially, if tunable to application scenarios are important when a wider penetration of the UC-technology is desirable into the daily-practice of protocol applications.

References

[1] M. Backes, I. Cervesato, A. D. Jaggard, A. Scedrov and J. K. Tsay, Cryptographically Sound Security Proofs for Basic And Public-Key Kerberos. *Proc. 11th European Symp. on Research. in Comp. Sec.*, 2006.

[2] M. Backes, B. Pfitzmann, and M. Waidner, A universally composable cryptographic library. *IACR Cryptology ePrint Archive*, Report 2003/015, January 2003.

[3] M. Backes and B. Pfitzmann, A General Composition Theorem for Secure Reactive Systems. *Theory of Cryptograpy Conference (TCC 2004)*, LNCS 2951, pp. 336-354, 2004.

[4] B. Barak, R. Canetti, J. Nielsen, and R. Pass, Universally Composable Protocols with Relaxed Set-up Assumptions. *In Proceedings of FOCS*, 2004.

[5] B.Barak, R.Canetti, Y.Lindell, R.Pass and T.Rabin, Secure Computation Without Authentication. International Cryptology Conference, Santa Barbara, California, USA, August 14-18, *CRYPTO 2005*, 2005.

[6] J. Camensisch, S. Krenn and V. Shoup, A Framework for Practical Universally Composable Zero-Knowledge Protocols, In: Lee, D.H., Wang, X. (eds.) *ASIACRYPT 2011*, 2011.

[7] R. Canetti, Security and composition of multi-party cryptographic protocols. *Journal of Cryptolology*, vol. 13, no.1, 2000.

[8] R. Canetti, Universally Composable Security: A New Paradigm for Cryptographic Protocols. *Cryptology ePrint Archive*, Report 2000/067. (received 22 Dec 2000, last revised 16 Jul 2013)

[9] R.Canetti, Universally Composable Notions of Signature, Certification, and Authentication. *17th IEEE Computer Security Foundations Workshop (CSFW)*, 2004.

[10] R. Canetti, Y. Dodis, R. Pass and S. Walfish, Universally Composable Security with Global Setup. *Cryptology ePrint Archive*, Report 2006/432. Nov 2006.

[11] R. Canetti and M. Fischlin, Universally Composable Commitments. *CRYTO 2001*, 2001.

[12] R. Canetti and J. Herzog, Universally Composable Symbolic Analysis of Mutual Authentication and Key-Exchange Protocols. *DIMACS workshop on protocol security analysis*, June 2004.

[13] R. Canetti and H. Krawczyk, Universally Composable Notions of Key Exchange and Secure Channels. L.R.Knudsen (Ed.): *EUROCRYPT 2002*, LNCS 2332, pp. 337-351, 2002.

[14] R. Canetti and H. Krawczyk, Security Analysis of IKE's Signature-Based Key-Exchange Protocol. In *CRYPTO 2002*, LNCS 2442, pp. 143-161, 2002.

[15] R. Canetti, Y. Lindell, R. Ostrovski, A. Sahai, Universally composable two-party and multi-party secure computation. *34th STOC*, pp. 494-503, 2002.

[16] R. Canetti and T. Rabin, Universal Composition with Joint State. *CRYPTO 2003*, 2003.

[17] S. Gajek, M. Manulis, O. Pereira, A-R. Sadeghi, J. Schwenk, Universally Composable Security Analysis of TLS. *ProvSec 2008*, pp. 313-327, 2008.

[18] J. A. Garay, P. MacKenzie, and K. Yang, Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, vol. 19, no. 22, pp. 169-209, 2006.

[19] J. A. Garay, D. Wichs, H-S. Zhouz, Somewhat Non-Committing Encryption and Efficient Adaptively Secure Oblivious Transfer. *CRYPTO 2009*, pp. 505-523, 2009.

[20] J.A. Garay, Y. Ishai, R. Kumaresan and H. Wee, On the Complexity of UC Commitments. LNCS Vol. 8441, *2014, EUROCRYPT 2014*, pp. 677-694, 2014.

[21] O. Goldreich, S. Micali, and A. Wigderson, How to play ANY mental game. In Proceedings of the nineteenth annual ACM conference on Theory of computing, *ACM Press*, pp. 218-229, 1987.

[22] J. Groth, Evaluating Security of Voting Schemes in the Universal Composability Framework. *ACNS 2004*, pp. 46-

60, 2004.
[23] F. B. Hamouda, O. Blazy, C. Chevalier, D. Pointcheval, D. Vergnaud, Efficient UC-Secure Authenticated Key-Exchange for Algebraic Languages. *Public Key Cryptography 2013*, pp. 272-291, 2013.
[24] H. Jayasree and A. Damodaram, A Novel Fair Anonymous Contract Signing Protocol for E-Commerce Applications. *International Journal of Network Security & Its Applications (IJNSA*), vol.4, no.5, Sept. 2012.
[25] D. Hofheinz and E. Kiltz, The group of signed quadratic residues and applications. In S. Halevi, editor, *CRYPTO 2009*, LNCS 5677, pp. 637-653, 2009.
[26] H. Krawczyk, HMQV: A High-Performance Secure Diffie-Hellman Protocol. *CRYPTO 2005*, LNCS 3621, pp. 546-566, 2005.
[27] Y. Lindell, Highly-Efficient Universally-Composable Commitments based on the DDH Assumption. *EUROCRYPT 2011*, pp. 446-466, 2011.
[28] H.Lipmaa, N.Asokan, V.Niemi, Secure Vickrey Auctions without Threshold Trust. *Financial Cryptography*, Bermuda, 2002.
[29] Y. Lindell, An Efficient Transform from Sigma Protocols to NIZK with a CRS and Non-programmable Random Oracle. *TCC (1) 2015*, pp. 93-109, 2015.
[30] A. Patil, On Symbolic Analysis of Cryptographic Protocols. *Master Thesis. MIT*, May 2005.
[31] C. Peikert, V. Vaikuntanathan, B. Waters, A Framework for Efficient and Composable Oblivious Transfer. *CRYPTO 2008*, pp. 554-571, 2008.
[32] Heiko Stamer, Efficient Electronic Gambling: An Extended Implementation of the Toolbox for Mental Card Games. C. Wolf, S. Lucks, P.-W. Yau (Eds.): WEWoRC 2005, LNI P-74, pp. 1–12, 2005.
[33] G. Ács, L. Buttyán, and I. Vajda. Provably secure on-demand source routing in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, vol.5, no.11, 2006.
[34] I.Vajda, A Universal Composability Framework For Anonymous Communications, *Journal of Computer and Communications Security*, vol.3, no.3, pp. 33-44. Sept. 2013.
[35] I. Vajda, Provably Secure On-demand Routing Protocols. *Pioneer Journal of Computer Science and Engineering Technology*, vol.6, no.1-2, pp. 19-39, 2013.
[36] I. Vajda, A proof technique for security assessment of on-demand ad hoc routing protocols. *International Journal of Security and Networks*, vol. 9, no.1, pp. 12-19, 2014.
[37] S.Walfish, Enhanced Security Models for Network Protocols. *PhD Dissertation*. New York University. January 2008.
[38] A. C. Yao, Protocols for Secure Computations (Extended Abstract) *FOCS 1982*, pp. 160-164, 1982.

Author's profile

István Vajda graduated from the Telecommunication Department at the Technical University of Budapest. He received the PhD and DSc degrees in 1985 and 1997, respectively. Since 1998, he has been a Professor at the Department of Informatics. He is the co-founder of the Laboratory of Cryptography and Systems Security (CrySyS). During 1990's his research interest was in algebraic code designs for secure multiple access channels. Recently, his research interests are in design and analysis of secure systems, with a special emphasis on provably secure cryptographic primitives and protocols. His application expertise covers secure wireless communication, secure routing and sensor networks.

How to cite this paper: István Vajda,"Can universally composable cryptographic protocols be practical?", IJCNIS, vol.7, no.10, pp.23-34, 2015.DOI: 10.5815/ijcnis.2015.10.03