The UC approach: an application view

István Vajda

February 15, 2013

Abstract: What kind of guidelines can the UC approach provide for traditional designs and applications? The aim of this report is to bring this theoretically rooted, computer scientist technology closer to the community of practitioners in the field of protocol designs.

Contents

1. Introduction

What kind of design and analysis advantages can the UC framework provide compared to the daily practice of protocol design and analysis? In this report, we try to follow a practical approach in order to

- present those technical tools rooted in the UC framework, which could enrich the arsenal of practical designers,

Technical University of Budapest E-mail: vajda@hit.bme.hu

- whet practitioner's appetite to dive more deeply into the theoretical foundations of the UC framework.

Though, a short technical report cannot set a goal to grasp the problem in its entirety, we try to touch relevant and convincing aspects to demonstrate the practical advantages of the UC framework. We give several examples for illustration of the presented concepts. Formal statements, conjectures and comments will also be shown to deepen the understanding of the application features of the UC framework. During these comparisons several related issues will be treated: among others, the relationship to the security game based definitions, the theoretical and practical importance of the realizability of the identification and authentication ideal functionality, an interaction-proof property, the technique of hybrids and modular design/analysis, the "technical loophole" of the simulation approach with respect to simulation failures, the importance of out-of-system physical security assumptions, the (Backes-Pfitzmann-Waidner) BPW's approach versus the Canetti's UC approach or the problem of time modeling.

We are not aware of papers with a goal of taking an effort to provide such a bridge. Obviously, we also undertake the possible criticism of too practical/informal or too formal/less practical from the two sides of this imaginary interface.

By lack of space, and by the size of the field, clearly, we have to refer the reader to the original formulations of the basic notions and definitions, we also use in this report. In this respect, surely the best and most authentic source is Canetti [11]. This paper provides all the needed definitions for the base universal composability (UC) framework. In addition, the key references from which this report profited the most are the following: the Joint UC (JUC) approach [13], the Global UC (GUC) approach [14], [24], the application/theoretical paper [12] on key exchange protocols, work [24] on realization theorems for authentication/identification protocols, paper [8] on secure computation without authentication as well as papers [4],[5],[17] on reactive security and universally composable cryptographic library from the Backes-Pfitzmann-Waidner's (BPW's) approach.

The structure of the report is the following. In Section 3 we step through the main elements of the UC framework in a concise way with special view to their design/analytic strength and application-oriented/theory-founding features compared to earlier traditional methods. In Section 4 we turn our attention to concrete design and analysis "guidelines" by considering the traditional and the UC methodology in parallel. In Section 4 we also treat the BPW's approach and the problem of time modeling.

2. Elements of the UC framework

In this chapter we step through the main concepts, model elements of the UC approach with a special view to the connections/independence to/from the traditional design. The main elements, treated below, are the following:

- definition and strength of the ideal functionality: simulation failures and resolutions, relaxation

- the practical/theoretical importance of the assumed level of adaptivity of the adversary
- assumptions on the security of the communication channels and their realizability
- setups, trusted functionalities
- instance separation and "stand alone" analysis

The UC model is adjustable along several factors (e.g. definition and strength of the ideal functionality, the assumed level of the adaptivity of the adversary, the assumed setup models and trusted functionalities, the availability of authenticated channels etc.), which strongly affects the realizability of a cryptographic task. A pragmatic corresponding question is the following: How to choose the set of these "factors" in concrete application problems and scenarios?
This problem and the corresponding answers have remarkable practical importance. If this set of conditions is not unique, then it provides room for cost/efficiency optimization.

2.1. Definition of the ideal functionality

What do we mean under a specific cryptographic task? Originally, it is informal, and the community has a more or less accurate/formal consensus on it. A formalized definition is a corresponding ideal functionality. This formalization is not unique and it is not a definition of "for now and ever". Especially, the level of abstraction has a strong effect on the demand against the cryptographic elements (e.g. primitives) and the setups by which they can securely be realized under different adversarial models (especially, at different levels of adaptivity).

2.1.1. Indistinguishability games vs. UC

A cryptographic primitive or a security mechanism can be specified in a variety of ways, such as a

a.) the traditional way of a list of properties that must hold in the face of attack,
b.) indistinguishability game against an attacker (*security by indistinguishability*),
c.) simulatability, based on the definition of an ideal functionality (*security by emulation of an ideal process*).

Note, all these three specifications may refer to the same cryptographic task and they are defined independently within their "framework". Therefore, it should not be a surprise, if incoherencies arise.

In Section 3 we will return to specification a.), in details. Now we concentrate on specifications b.) versus c.).

The game based security relies on the indistinguishability paradigm, often called indistinguishability games. UC security also applies this paradigm within the simulatability approach.

In approach b.), the adversary is allowed to access a corresponding oracle during a learning period before he comes ahead with an attack. During this learning the adversary is allowed to send requests to the oracle independent (in a corresponding sense) from the target to be attacked finally. The oracle answers should not provide the adversary with information which increases its attack success non-negligibly, compared to the case, when it has no access to the oracle.

Basically, from the reason of easier abstraction in the definition of ideal functionality, the typical case is that specification c.) is stronger then b.), i.e. if a realization is secure by spec. c.) then it is also secure by spec. b.).

It may happen that we can *"convert" the game into a security definition within the UC-framework*. Assume a cryptographic task, the security of which is defined both as an indistinguishability game and also in UC-framework, i.e. an ideal functionality is defined in the latter case. We would like to bridge the two security definitions, typically by showing that a single session protocol is secure according some UC definition if and only if it is secure according to the game-based definition (referring typically, to some multi-session extension of the protocol).

The game based security definition is rephrased in the UC framework. The oraculum of the game is included into environment Z. Z carries out the game. The adversary of the game requests Z to invoke instances for learning and also Z provides the target instance for the attack (may be together with concurrent instances by the wish of the adversary). Environment Z carries out the job of distinguishing by observing the communication at the global interface. Finally, Z outputs the corresponding binary decision.

For example, such a method has been used in [12], where the authors established an "if and only if" bridge between the game based and UC-based definition of session key exchange.

Example 1 *(anonymous communication):* The authors of [15] proposed a computational indistinguishability approach, similar to the definition of semantic encryption schemes, in order to give a strong definition for anonymity under computational constraint (ind-anonymizer):

"…the adversary produces two message matrices (which encode message senders and receivers in a standard way), and it is allowed to passively observe the execution of a communication protocol under a random one of these two matrices and then is required to have non-negligible advantage in determining under which of the two matrices the protocol was executed."

In [22] we prove that our ideal system F_{acom} provides an equivalently strong definition for anonymity with the significant advantage that F_{acom} is also part of a proof system for assessment of anonymity provided by different realizations. In particular, it was shown that under global passive adversary an anonymous communication scheme Q is an ind_anonymizer if and only if protocol π_Q UC-realizes ideal functionality F_{acom}. The result was extended also to adaptive case.

□

Example 2 (*public key encryption, digital signatures*):

The ind-CCA secure public key encryption as well as digital signatures secure against existential forgery are equivalently UC-secure against static adversaries [11].

Here we assume that different concurrent instances of the primitives choose independently generated keys. If a set of parties within the run of a protocol instance uses a public key encryption (a digital signature) primitive with fixed keys for several times then ind-CCA (existential forgery) ensures secure separation assuming that the calling protocol instance takes care of keeping the corresponding plaintexts (documents) different (e.g. by inserting appropriate nonce values). Note, keeping the input to the "common" functionality different, is analogue to the JUC (Joint state UC) technique (we return to JUC in subsection 2.5).

For an example of application, we refer to [18] and [19], where such primitives were used in realization of several different secure routing tasks.
□

2.1.2. Simulation failures and solutions

In nutshell, the UC approach is a security assessment where security is defined by emulation of an ideal process. The real power and unique advantage of the method is the assurance of secure composition with arbitrary protocol. Technically, the essence of the security proof is the construction of the simulator, usually, a black box simulator.

It may happen, that we construct a seemingly secure protocol for a cryptographic task, however, we fail to carry out the simulation (excluding the case, when the simulation can be done just we failed to find it out). The proof may stuck by a *commitment problem*, when at some step within the run of an instance the simulator has to produce a real value *c* in vain of some necessary information *d* (e.g. secret key, plaintext etc.), such that value *c* remains acceptable for a PPT distinguisher even when the missing information *d* becomes known at some time later.

Obviously, if information *d* uniquely determines value *c* then this problem cannot be resolved. This is the case of deterministic functions, for instance, usual hash functions, where the simulator commits to a hash value *c* without knowing message *d*.

However, if simulated value *c* is a (realization of a) random variable with a p.d. which cannot be distinguished by a PPT distinguisher from the real value (real random variable) even after getting access to *d*, then the simulator solves the problem. Indeed, the probabilistic approach and the corresponding indistinguishability tasks are at the heart of this simulation-based definition of security.

Taking a general view of this simulatability problem, at one side of extremes, we find the deterministic function (e.g. deterministic hash function), on the other side the random function (e.g. public random oracle). In the middle, we find pseudorandom functions (e.g. semantically secure public key encryption function). The most randomized functions (n bit to n bit random functions with uniform distribution over the total output space) provide trivial possibility for simulation; without knowing the input we can choose randomly any n bit long string.

The two main causes of simulation failures are the following:

- a too strong ideal functionality (e.g. an ideal functionality corresponding to usual hash functions [6] or key exchange functionality [12])
- adaptivity (especially, full adaptivity) of the adversary

The first problem with a strong ideal functionality can be remedied by the relaxation of the definition. This may mean, for instance, leaving a fully symbolic deterministic model and allowing tolerated impairments or supporting the simulator with carefully chosen additional information.

In the case of adaptive adversary the simulation process is divided in time by a corruption event into two stages: during the first stage honest protocol messages has to be simulated without having access to the corresponding secret/unknown stuff, while after corruption all protocol messages simulated in the first stage must remain consistently looking in the view of a distinguisher after getting access to secret/unknown information (e.g. for a black box adversary in base UC, or for the environment, in case of shared functionalities in GUC).

Relaxation of the ideal functionality

A completely symbolic ideal functionality provides no additional information about the content of a message to the adversary just as much as it was known by it a priori (i.e. the space from which the element may come from, the bit length of the message or of its parts, etc.). Here the knowledge of the adversary (the simulator) of the ideal system about an "observed" element is just that it is an arbitrary element from the space of ciphertexts, plaintexts etc. With such knowledge, we cannot expect that the simulator is able to simulate the corresponding actual real message which comes from a subspace according to a probability distribution *D* unknown by the simulator. In other words, according to its knowledge, the simulator's best action is a uniformly random selection from the total space, which if it can be distinguished from a sample from *D*, the simulator fails to accomplish the actual simulation.
We have to help the simulator to make him possible to access random samples from distribution *D*. Such a help can be provided by primitives with appropriate properties, like non-committing public key encryption [9] or adding an appropriate interface to the ideal functionality like a non-information oracle [12].
In order to provide UC-security against adaptive adversaries we have to use such methods.

Example 3 *(non-information oracle)*: The ideal functionality of key exchange F_{KE} ([12]) was too strong to allow simulation in UC framework and the authors weakened this ideal functionality by adding a so called non-information oracle to it. The authors referred to the problem as a "technical loophole" of the general approach of security by emulation of ideal functionality.
Here we propose a little bit different view of this problem:
Rather, F_{KE} was too strong with respect to modeling the capabilities of the adversary appropriately, in other words, with respect to tolerable imperfections. The information gained by - even a passive - (real) adversary is not only the bit-lengths (the length of a ciphertext), because he sees a sample from a probability distribution *D* determined by the corresponding cryptographic primitive (which is more than assuming a uniform distribution over the total space of ciphertexts).

The ideality should not be expressed by denying access to information about distribution *D* and denying just the access to any information about the secret information underlying the sample of distribution D (by PPT algorithms).

Such an event is the result of the not-comparability of specifications by the two approaches, in particular, the strong ideal functionality and it is not a

consequence of some inherent technical problem ("weakness") in the simulation approach.
□

It has been mentioned above that deterministic functions (like usual hash functions) cause simulation failure in corresponding commitment cases. The next example refers to randomization together with relaxation of ideal functionality in order to resolve this simulation problem:

Example 4 (*hash function*): In [20] the following ideal properties i-ii.) of hash functions were relaxed by verification property iii.), where
i.) Ideal collision freeness: $r\_hash(m) = r\_hash(m') \rightarrow$ m= m′ *for all* m, m′,
ii.) Ideal secrecy: If we have a new hash value *h* of message *m* and we want to find information about the hashed message we cannot do better than forgetting about *h* and just relying on the verification oracle I_m by invoking it with guessed messages polynomial many times,
and
iii.) Verification capability: Only those users are able to verify (re-calculate) a hash value in the knowledge of the corresponding message, who are included in a set (*Vset*). *Vset* is determined by the sender of the hash value.

This ideal function can be realized in the standard model of cryptography (in particular, without relying on random oracles):
(Theorem 2, [20])**:** The symbolic system with ideal *r-hash* model is securely implemented in the real system with the real *r-hash* primitive (Definition 2. [20]) in the sense of BRSIM/UC in the standard model of cryptography, assumed that honest users authorize only honest users to carry out verification.
□

Sometimes, we have to incorporate tolerable imperfections into the ideal functionality, where tolerable imperfections model those attacks that are unavoidable, or too costly to defend against, and hence, we tolerate them.

Example 5 (*tolerable imperfections*):
In case of *anonymous communication* [22], if actually there is only a single active sender and its transmission cannot be hidden (tolerable impairment) then sender-anonymity, obviously, cannot be guaranteed for a message on the corresponding communication link and without dummy packets it cannot be guaranteed at any other point of the system observable by the adversary. Indeed, the anonymity requirement must be coherent with the set of tolerable imperfections.
□

Different *setups* included also into the ideal system can also be considered as a supplement to the ideal functionality. This way the potential for realizability of cryptographic tasks are increased even against strong adversaries, in particular, a full adaptive adversary. In this class of setups we find practically/theoretically useful shared functionalities e.g. CRS, ACRS, KRK etc.

Special primitives

Technically, the essence of security assessment in UC framework, is to find an appropriate simulator. A black box simulator translates abstract objects handled by the ideal functionality into real bit-strings for the real adversary. In the view of the real adversary such a simulated view should be indistinguishable from the real one he sees in the real system. The interesting problems for such "translation" are those when the simulator is in vain of information and as a consequence, it gets stuck in a commitment problem. Informally, the ideal functionality describes the guarantees in "information theoretical" terms (like a party knows, does not know or partially knows some information) and the question is the ability to reproduce these security guarantees in a PPT environment. The success of this "translation" effort is supported by special primitives.

Example 6:

Passive adversary:
A typical case of simulation failures happens when the simulator has to produce a ciphertext *c* without knowing the corresponding plaintext *d*. If the (real) adversary has no access to the corresponding decoding key, a *semantically secure (ind-CPA) encryption* will do the task: this is the technique of public key *encryption of dummy plaintexts*. If, sometimes later, plaintext *d* becomes known the real adversary will not see the simulated ciphertext inconsistent. This is the case, when protocol parties are honest and the (real) adversary is passive (i.e. sees the bits transmitted over – non-private - communication channels). However, if during the run of the instance or later on, the party, in the name of which the simulator produced the simulated ciphertext, becomes corrupted and the adversary gets access to the decoding key, the above simulation trick collapses: the adversary trivially compares the dummy plaintext to the real one.

If the parties are trusted to erase records of their states, even adaptively secure computation can be carried out using *known primitives*. However, this total trust in parties may be unrealistic in many scenarios and then special primitives are needed.

Adaptive adversary/UC:
In the (full) adaptive case, it makes a big difference of how the adversary/ distinguisher gets access to the secret stuff during the run of the instance:

If it happens only via the mediation of the simulator, then the simulator may cheat, relying on the approach of indistinguishable random substitute. This is the typical case in base UC setup. Probably, the best example is the *non-committing encryption* [9], which makes possible that after a corruption event, the simulator becomes able to come up with a decoding key consistent with the simulated ciphertext and plaintext.

Adaptive adversary/GUC:
However, if even the environment (distinguisher) Z has direct access to the secret stuff within a corrupted element, like in the case of corrupted shared functionalities, then considerably less room remains for disguising tricks for the simulator. Clearly, there remains no room for tricking with substitution of the corresponding real secret key similar to mentioned above.

The special case, when the protocol for a cryptographic task needs only encryption operation without the need for decoding by using a secret decoding key, allows some room for successful simulation even in case of GUC and even in case of adaptive adversaries. Such a cryptographic task is a bit commitment problem, where the "decoding" operation is the opening of the commitment, when the committer simply shows up a simulated committed value. The simulator prepares to a corrupted verifier by producing commitment with – so called - equivocal property. In general, *equivocality* means that a simulated object can fit to different input values (or sets of such vales) indistinguishably. An example for a corresponding primitive is a public key encryption which maps into the space of ciphertext indistinguishably from a mapping and produces randomly chosen ciphertext (PRC-CCA, *Pseudo-Random Ciphertext,* [7]).
□

Impossibility and realizability results

An important advantage of the UC framework is that in this rigorous model general impossibility/realizability results can be proved. At first glance, it might seem that such results are at the far end of theoretical interest, however, it is not so: these results have considerable practical value:

An impossibility result says that under the corresponding assumptions (model scenario) an UC-secure realization is not possible. The model scenario consists of the security assumptions about the underlying communication channels (e.g. raw channel, authenticated channel, secure channel), assumed setups and trusted third parties, assumed ideal algorithmic components (subroutines) and the level of adaptivity of the adversary. For instance, in the assumed setups, "out-of-system" pre-distributed shared keys or other physically supported security ingredients are also included. Different sets of assumptions also mean different cost and efficiency/complexity. From practical point of view all these components could be traded-off.

2.2. Adaptive adversary

The UC approach essentially exceeds the capabilities of traditional approaches in the fine grade modeling against adaptive adversaries. As it was already mentioned above, the complexity of the design, and the analysis as well as the complexity of primitives for secure realization jumps when we switch to full adaptivity, especially, together with shared functionalities (causing dependent states for concurrent instances).

Therefore in concrete applications we have to analyze the level of adaptivity of the imagined adversary carefully, together with the possible non-algorithmic procedures to limit or decrease the necessary level of adaptivity.

Example 7 (*cost of adaptivity*): Achieving UC/GUC security against full adaptive adversary, typically, costs a lot in terms of complexity/efficiency (if we can show a proof at all). Facing such a task, we should think it through thoroughly if an adversary is really able to attack during the time window of an instance in an adaptive way with non-negligible probability.
□

Example 8 (*quasi-adaptivity*)**:**
Papers [18], [19] provide the first provably secure routing protocols in the UC framework. The proof uses the composable cryptolibrary of Backes, Pfitzmann and Waidner [4].

For instance, consider the case of source routing: the initiator of the route discovery process generates a route request, which contains the identifiers of the initiator and the target node, and a randomly generated request identifier. Each intermediate node that receives the request for the first time appends its identifier to the route accumulated so far in the request, and re-broadcasts the request. When the request arrives to the target node, it generates a route reply message. The reply message is sent back to the initiator on the reverse of the route found in the request.

Here we propose the notion of *quasi-adaptivity*, illustrated for this application. We restrict the full adaptivity by assuming that when certain subroutines are running the adversary does not carry out adaptive attacks. In the application example, we assume that the adversary which does not adapt during the run of an instance of a cryptoprimitive (e.g. digital signature) run by a node; it has to make its decision about the corruption of a node before the node starts processing an incoming protocol message. Assuming such an adversarial model, we allow adaptivity and at the same time we can use non-special primitives, which are need to be secure only against a static adversary.
□

The task of identification and authentication

An authenticated channel provides double security services: integrity protection and party identification. These two services are independent; both of them can be realized without the other. We can share secret keys with a party without knowing his identity and using these keys we can realize integrity protection. However, if these secret keys are pre-assigned by some trusted out-of-system approach, we might know the identity of the party, we share keys with. In this latter case, the success of integrity protection, implicitly, provides also identification. In general, secure identification assumes a trusted third party which provides some kind of registration service.

Almost all known results for UC-secure realization of different cryptographic tasks assumed (were proved in) an F_{auth}-hybrid model. Assuming the existence of authenticated channels can be visualized as follows: two remote parties, who "personally" know each other, "talk" as if they were within line of sight and hearing distance, potentially together with the adversary within line of sight and hearing distance. Such an elimination of remote communication illustrates its independence from the core of the particular task, which - in the above analogy – is a security problem the "talking" parties, who may not trust each other, want to solve.

A results in [24] shows that F_{auth} can only be realized against static adversaries (under pre-assigned shared keys or under F_{krk}- hybrids). It follows that physically authenticated channels has to be available, if we want UC-security against full adaptive adversaries:

We cannot eliminate or get around the out-of-system physical/procedural security assumptions.

The informal reason is the following: here we have to work directly over bare channels, and therefore the secret keys has to be deployed in straightforward way via authentication checksums to protect the transmitted bits against manipulation by an adversary. Note, a receiving party has to be able to check the authenticity of each message it receives – even the first -, and the only possibility for the transmitting/ receiving party is to use secret information.

Secure identification is implicit in authenticated channels. Some kind of registration service includes the out-of-system procedures of secure identification; let it be, for instance, a communication node or a person. The point here is that there always exist out-of-system physical/procedural components which cannot be eliminated or substituted by pure algorithmic methods.

By a *conjecture of Walfish* ([24], p.103), it is "impossible to realize $\mathrm{F}_{\mathrm{auth}}$ with a forward secure protocol even if erasures are allowed". Here we return to this conjecture.

Proposition 1: In the model defined above, we confirmed the conjecture of Walfish ([24], p.103) about the impossibility of forward secure realization of $\mathrm{F}_{\mathrm{auth}}$ even if erasures are allowed.

Proof: (Sketch)

Because, F_{id} can (trivially) be realized over $\mathrm{F}_{\mathrm{auth}}$ -hybrid, it is enough to consider the conjecture for F_{id}. Consider the following (real system) model 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/key *k*, generated by algorithm *KeyGen*.

Assume a two-party identification scenario between parties *B* and *C*, where party *B* authenticates to party *C*. During this authentication, party *C* receives (in one or more steps) bit-string $I(r_B, r_C, k_B, pub)$ (=*i*), where algorithm *I* is public; r_B and r_C are temporary 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.).

C runs a public verification algorithm *V* with input (I, r_C, *pub*). Assume, after the run of an instance, honest parties erase temporary internal state elements, here, random elements r_B (r_C). The adversary gets access to value *i*.

Assume (*KeyGen*, *I*, *V*) provides static security. In particular, to support the simulation of honest party *B*, we assume, that the distribution of samples $i_1 = I(r_1, r_2, k^{(1)}, pub)$ and $i_2 = I(r_3, r_4, k^{(2)}, pub)$ are indistinguishable for a distinguisher, where $k^{(1)}, k^{(2)}$ are two different outputs of *KeyGen,* (r_1, r_2, r_3, r_4) are independent random input elements and all these inputs are unknown by the distinguisher. Assume, however, if in the above task, the distinguisher gets access also to key $k^{(1)}$, he becomes successful with non-negligible probability.

Having defined the real system, assume, the adversary compromises 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 gets access to the real key, he is able to distinguish a simulated transcript from a real one. Note, there should exist such a distinguishing algorithm, otherwise, anybody could impersonate the party successfully without knowing the secret information.

□

2.3. Setups: theoretical or practical?

Circularity of assumptions

Without appropriate setup assumptions most cryptographic tasks are unrealizable. There are practical and theoretical setup models. For example, PKI/ KRK (Key Registration with Knowledge) and "weak" PKI ("bulletin board", "bare public key") models can be considered practical. However, the primary role of CRS and ACRS setup models is to find the minimal setups under which most cryptographic tasks can be realized against certain level of adaptivity of the adversary, and these setup models are first of all of theoretical interest.

The "simplest" setup is the CRS (Common Random String) [7]. In the UC framework, parties of an instance are able to obtain a common random string chosen from a distribution *D*, which is public in the sense that it is known even by the adversary; however, it is not available for other instances. In the GUC framework, it is known also by the environment, which technically means that the string must be the same in the real and the ideal system.

If we are thinking about realizability of such a setup, one of the problems is the guarantee for authenticated channels between the parties of the instance and the setup: indeed, realization of an authenticated channel itself needs a setup, stronger than CRS. Resolution of such circularity of assumptions can only be done by involving out-of-system methods "to cut the circle", for instance, physically ensuring the existence of such channels. In this sense, the use of such a theoretical approach is questionable if finally we want to see a realization fully over bare model. We can arrive to similar conclusion with the stronger theoretical setup, the ACRS [14].

Dependence of protocol layers via setups: an internal shared functionality model

In global UC, instances of the same or different protocols may have dependent state via the shared setup functionalities. Here we propose *an internal shared functionality model.*

Consider the case when different layers within a protocol would like to use common functionality (a shared instance of it). The composition technique (i.e. freely "plugging in" a UC-secure subroutine realization) is supported if the corresponding layers call independent instances of the shared functionality (e.g. independent pre-assigned keys, independent instances of cryptographic primitives). Independence at different layers is a clear cut assumption. In spite of that, here arises the following question: Could we use a "master" functionality, which could serve different layers (like a JUC functionality serving several instances of a protocol)?

Consider a protocol instance with two layers and with common functionality (Fig.1). We want to plug-in the ideal or the real subroutine interchangeably into the lower layer, i.e. in an indistinguishable way in the view of the upper layer. Because both layers have access to a common functionality, this scenario resembles the global UC setup, where the environment (here played by the upper layer) has access to a shared functionality both in the ideal and the real system (which models that arbitrary protocols can use a functionality PID-wise).

To strengthen such an analogy, imagine an attack scenario and implementation circumstance, where an adversary is able to launch attack against a protocol by HW/SW modules.

Following the GUC-analogy, we might require that a realization of the lower layer should be "GUC-secure" by the actual common functionality/setup. The advantage of such a relationship, in general, could be that, we might profit from those analogous results, e.g. realisability/impossibility results.

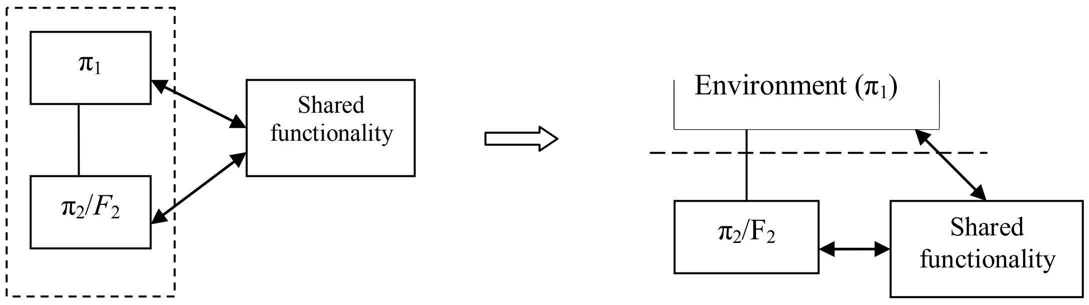

Fig.1: An internal shared functionality model

Proposition 2: Internal shared functionality model is proposed, which provides the potential to apply known results of GUC-approach in the analysis of shared-functionality dependent protocol layers.

2.4. Instance separation

An UC-secure realization of a cryptographic task offers

- secure *separation of instances*
- *simulatability* of the message flow for honest and corrupted cases during the "stand alone" analysis

Instance separation means the interaction-proof property between the target instance and other instances of the same or other protocol: a corresponding attack by an adversary results in the abort of the target instance sometime during the run. Once the instances are securely separated, the analysis is reduced to the examination of the target instance, where the adversary is restricted to use only the information it can get from the messages of the target instance together with the state of parties in case of corruption.

Separation plane

We can visualize a "separation plane" as an *X x Y = concurrency x time plane*, with

X axis: separation from concurrent instances of any protocols,
Y axis: separation from past instances of the same protocol.
Along the *Y* axis we can consider attacks in both directions in time:

- from past to present (e.g. CCA-security, existential forgery),
- from present to past (forward security)

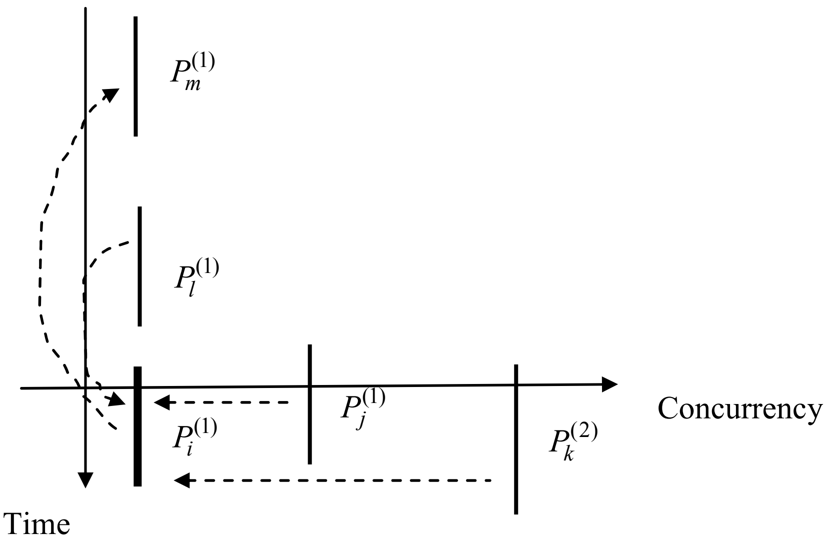

Fig. 2: Concurrency x time plane
($P_n^{(m)}$ is the n-th instance of protocol $P^{(m)}$; $P_i^{(1)}$ is the target instance)

An interaction-proof property of protocols

By the paradigm of secure emulation an adversary is successful if he is able to distort the output in a distinguishable way; the (total) output of the implementation as a random variable is different from the same in the ideal system. Consider a *conscious adversary* with an aim, which is not just distorts the output in arbitrary way, but in some controlled way. Such a control is modeled by a general relation *R* on *U*x*U*, where *U* is the output space. Such an adversary is successful, if there exists a relation R, such that, the adversary is able to distort the output, according to R, with non-negligible probability.

Report [23] suggests a game-based definition for *interaction-proof property* of protocols. The best known attack type against secure separation of instances is the so called interleaving attack. Assume a general cryptographic task and an implementation π (see Fig.3). Assume, that a restricted, stand alone attack against π cannot be successful in the sense of emulation of an ideal process (shortly, protocol π is "stand alone secure"). Here, the stand alone assumption is meant over the total separation plane, i.e. the adversary has no access to any information, in addition, what can be exploited from a single instance, neither from past nor from concurrent instances. Our intention with this model is to exclude that an implementation intended to be used in public network environment is so weak that it is not able to achieve the security goals even when it is running completely separated from the outside world. This assumption is modeled by ind-secure realization of the ideal functionality by a stand alone system (relationship (2) in Fig. 3).

The game-based definition is proposed for *interaction-proof property* of protocol π corresponds to relationship (1) in Fig. 3 ("R-secure"). This relationship is expectedly more stringent than ind-security followed by the UC approach, i.e. in general, it requests stronger security guaranties from realizations.

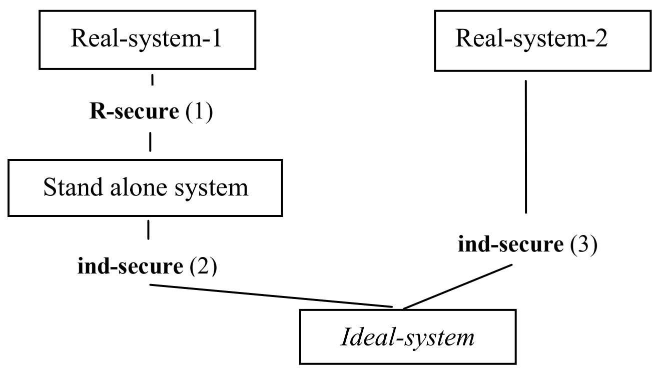

Fig. 3: Interaction proof property

Conjecture 1: *Consider a "conscious adversary" and a protocol π, which is interaction-proof (by definition of [23]) as well as "stand alone secure". We expect, by this way, we can define stronger security guaranties while keeping the definitional strength of the ideal functionality.*

JUC

In case of JUC, the same instance of a shared functionality serves several instances of the same protocol: instances accessing the common functionality (in the ideal model) are separated by *ssid*'s (sub-sid). These *ssid*'s are prepended to the input sent by the instance to the instance of the common ideal functionality (having session identifier *sid*). In other words, the inputs sent to the common functionality are forced to be different. For example, JUC encryption/digital signature realization for message *m* is done by encryption/digital signature on extended message *m'*=[*ssid,m*].

By this way, the ideal functionality implies that any realization (of this common functionality) must be able to separate messages securely with different *ssid*'s (shortly, "*JUC-secure"* primitive). On the other hand, in security game definitions of cryptographic primitives we require the requests (inputs) to the oracle to be different (from each other and from the target) during learning/testing actions. Let this distinction of inputs be realized by a unique string (like an *ssid*) prepended to the message.

A further parameter is also underlying both these approaches: we can distinguish different "attack classes" reflecting the strength of the adversary, like CPA or CCA attacks. Shortly, we refer to the security by game approach (per attack class) "*game-secure*". The (JUC) approach is analogue to "game-secure" solutions, when we consider the secure separation along the time (*Y*) axis (Fig.2).

Proposition 3: "Game secure" primitives provide "JUC-secure" implementations (per attack class).
Proof: Straightforward from the meaning of "game secure" and "JUC-secure" detailed above.

3. The traditional sound design guidelines for protocols and the UC approach

The general questions for this chapter are the following:
Can traditional sound design guidelines for protocols guarantee an UC-secure realization of a cryptographic task? What an extent, if not completely?

Recall the traditional design guidelines comprise the following main issues:

- definition of a security goal (e.g. "which party what will know at the end of the run")
- adversarial model (passive, active etc.)
- universal guidelines (because of such universality, these guidelines are expected to support security of realizations also in the UC framework)

3.1. Security goals

Example 9 (*traditional security services of key exchange*): The set of security features of traditional key exchange protocols, for setting up a fresh symmetric session key, are the following: implicit key authentication, key confirmation, explicit key authentication, freshness of keys.
"Implicit key authentication" means that a party can be sure that only the assumed remote party (and also the trusted third party if there is such) can have access to the key. "Key confirmation" means that the remote party confirms the possession of the new session key. "Explicit key authentication" is "implicit key authentication" together with "key confirmation". "Key freshness" means that a party is sure in the freshness of the key.

In comparison, consider ideal functionality F_{KE} ([12], Fig.7, p.29):

- F_{KE} guaranties (in the above terminology) explicit key authentication and key freshness
- in addition, F_{KE} defines the case of corruption: if corruption happens during the run of the instance (neither party is aware of the session key, yet) then the adversary sees the session key, otherwise, i.e. if corruption happens after the run has been finished, the adversary obtains no information about the key (forward security)

□

The traditional approach of security definition of a task requires *correctness* and *secrecy.* Correctness means (in terms of simulation based approach) that at the interface of honest users the outputs in the real and the ideal system are indistinguishable (S_1 in Fig.4). Correctness is connected to the security goal approach of traditional specification (usually defined by a set of security requirements). Secrecy means that the adversary does not learn anything more in addition to the information it can obtain from corrupted parties (S_2 in Fig.4).

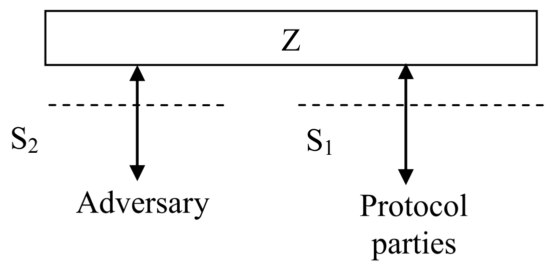

Fig.4. Interfaces to the environment from the adversary and the protocol parties

In contrary, in the UC approach, the so called *global output* ($[S_1|S_2]$) is considered, which is a random variable that consists of the concatenation of the outputs of honest parties and the adversary (compromised parties are incorporated into the adversary). The point is the following: security goal may be fulfilled without the indistinguishability of the global interface. This implies that security provided by the UC-approach is stronger than the traditional.

Example 10 *(security goal vs. emulation of global output)***:** Consider a session key agreement protocol between parties *B* and *C* over secure channel:

1. $B \rightarrow C$: k_1
2. $C \rightarrow B$: k_2
3. B, C: $k=k_1+k_2$

where k_i is binary n-bit string and "+" addition is XOR.
S_1: Correctness: honest parties output the "correct" value, i.e. the XOR of the random n-bits generated by the parties (honest or corrupted).
S_2: Secrecy: (in this case trivial); the adversary (passive/static) does not learn anything more from the run than what can be learned from the information received by the corrupted parties (inputs/received messages).

The adversary freely communicates with environment Z, so at interface S_2, we can assume that the adversary outputs the random n-bit, which a corrupted party "generates". Say, party *C* becomes corrupted and the adversary outputs k_2 at interface S_2. The honest party outputs *k* at (service) interface S_1 (e.g. key exchange is a lower layer in a protocol, which serves an upper layer). Consider the random variables *U* and *V* at interface S_1 and S_2, respectively, i.e. in our case $U=k$ and $V=k_2$.

If the adversary passive or static, then random variables *U* and *V* are independent: $P(U=u \mid V=v) = P(U=u) = 1/2^n$. However, if we assume an adaptive adversary, he is able to set session key *k* to an arbitrary value by choosing k_2 appropriately. For instance, by setting *k* to all zero bits: $P(U=0 \mid V=k_1)=1$.

For a fair comparison, recall, that in traditional approaches only passive adversary and at most static corruption used to be assumed.

□

3.2. UC analysis of designs and the technique of hybrids

The most important advantage of the approach of "security by emulation of an ideal process" is the *composability*; i.e. the guarantee of secure functioning of an UC-secure element when it is invoked by arbitrary protocol. A closely related design and proof technique is the technique of *hybrid protocols*. The technique of hybrid protocols can be considered as a decomposition of the original task into sub-tasks, where sub-tasks can be as granular as cryptographic primitives.

Example 11 (*chain of hybrids/sequential composition*)**:**
Sometimes, bigger protocols have a serial structure, where the protocol can be decomposed into a chain of parts corresponding to sub-tasks, where each sub-task in this chain assumes the security of the previous elements along the chain. For example, in case of a traditional protocol for secure two-party communication:

1. Initialization (pre-assigned information/setup) - F_{init}
2. Party authentication (A) - F_1
3. Key exchange (K) - F_2
4. Secure channel (S) - F_3
5. Secure communication (C) - F_4

$$F_{init} \rightarrow \text{A}/F_{init} \rightarrow \text{K}/(F_1, F_{init}) \rightarrow \text{S}/(F_2, F_1) \rightarrow \text{C}/F_3$$

The aim is to break the analysis of the protocol into the analysis of the element of the chain of hybrids.
□

Looking at the chain of hybrids in Example 11 one might feel an analogy from the probability theory: the conditional probability. Let's consider just the interface between the protocol parties and the environment, where parties send output only as the last step of the protocol. This output, modeled as a random variable, is determined by the local random elements of parties, the algorithm of the adversary (and its random elements) as well as the random variable describing the communication with the subroutine(s) (ideal functionalities). In this respect, the probability distribution of the output depends on the subroutine(s). The chain of hybrids in Example 11 is an analogue to the conditional distributions, where in the conditions we find (the shortest list of) those hybrids which determine (dependent with) the considered sub-task variable.

3.3. Design guidelines

The traditional (informal) sound protocol design guidelines (e.g. [1]) usually emphasize the
(a) clear definition of security goals/requirements (Principles 4-9 in [1]),
(b) explicit assumptions (Principle 2 in [1]),

as well as they suggest several simple

(c) techniques for instance identification and separation (Principles 1,3,10 in [1]).

Informally, we find obvious connections to these guidelines from the UC framework:
(a') definition of the ideal functionality for the considered task,
(b') definition of adversarial capabilities (adaptive, non-adaptive…), communication network (authenticated, private,…), setups/trusted parties
(c') an inherent feature of the UC approach is the separation of instances in connection with the "stand alone" analysis.

We considered the comparison (a-a') in subsection 3.1, with conclusion that the UC approach is more general; the traditional approach (a) is a special case. As for issue (b), in the UC approach the system setup is rigorously formalized not just as "static list" as in (b) but via precise definition of dynamic, "runable" functionalities.

In [1] we find the following guidelines (reformulated below):
- The (intended) meaning of each protocol message should be clean-cut (the content of each message should represent the (intended) meaning completely).
- If the identity of a party is essential to the meaning of the message, then the identifier of the party should appear explicitly in the message.
- It should be uniquely decidable, which instance the message is corresponding to.

In the UC framework, in the ideal system the general structure of a message is the following:

(*message/command type, sid, "payload"*),

e.g. (*sign*, *sid, m*). *sid=(A,B,…sid')* , where *A*,*B*,… identifies the parties. Each message is identified uniquely in the ideal system. A secure realization must guarantee such unique separation. (Note, this does not necessarily mean that, for instance, in the real protocol each and every message must contain similar *sid* element/structure).

Before we show examples for these design approaches, we recall two important techniques for message/instance separation:

"Sequence numbers"

When a *flow of messages* is transmitted, the traditional approach is to use sequence numbers to separate consecutive messages within a protocol instance (usually, together with techniques preserving the integrity of the flow).

In the UC approach, an instance serves the transmission of a single message only. If the (natural) realization uses a keyed cryptographic primitive, then from cost/efficiency reasons we use the same key (at least) for all messages of the flow. The solution is to assign a *sid* to the flow (the "traditional" instance), and sub-sid (*ssid*) to the single message instance. The actual cryptographic operation realizing such multi-session extension of the ideal functionality, should ensure the secure separation of the messages of the flow. (Note, *ssid* in itself does not imply the integrity preservation of the flow, it just distinguishes the messages. In this respect, it is not a sequence number, however, a realization could use sequence numbers, which supports also integrity protection.) We shall return to the JUC approach also in the examples.

"Independent keys per applications"

Almost the best known traditional security guidelines refer to the secure erasure of expired secret keys as well as the selection of independent keys for different protocols (usually, phrased as different applications):

Example 12 (*implicit authentication*):
Implicit authentication was an insecure realization practice for traditional session key exchange protocols: the long term encryption key ("terminal key") is intended to provide also authentication for the transmitted fresh key by inserting fields into the message, the content of which provides known redundancy for the receiving party. It may happen that an actual realization is secure, however, such an approach is not sound: independent keys have to be assigned to the tasks of encryption and to the authentication. Specifically, in this case, the encrypt-then-MAC paradigm [16] should also be adhered to.
□

Example 13 (*session key exchange*)**:**
The task of session key exchange is considered in this example. The modular design and analysis approach is demonstrated. The session key is transmitted over a secure channel. Secure messaging is built upon initially distributed keys. The initial key can be a long term pre-assigned shared key or public key (e.g. relying on $\mathrm{F}_{\mathrm{krk}}$ setup). Secure messaging can be realized in one or two steps. At first, the one-step case is shown:

$\mathrm{F}_{\mathrm{init}} \rightarrow \mathbf{SMT}/\mathrm{F}_{\mathrm{init}}$ **:**
Canetti and Krawczyk [12] show a "weak" UC-secure realization of secure messaging against adaptive adversary over $\mathrm{F}_{\mathrm{init}}$, where initialization means pre-assigned long term shared keys. The weakened version of ideal functionality F_{KE} relies on a non-information oracle (recall, weakening resolves the commitment problem, which arises, when corruption happens). Semantically secure (ind-CPA) symmetric key encryption and Message Authentication Code (MAC) secure against chosen message attacks is used by following the encrypt-then-MAC paradigm [16].

Now, consider the case, when an authenticated channel is realized first, followed by realization of secure messaging over $\mathrm{F}_{\mathrm{auth}}$-hybrid:

$\mathrm{F}_{\mathrm{init}} \rightarrow \mathbf{AUTH}/\mathrm{F}_{\mathrm{init}} \rightarrow \mathbf{SMT}/F_{auth} \rightarrow \mathbf{KE}/F_{smt}$ **:**
Here, we consider a pure symmetric key approach. Pre-assigned shared long term keys are assumed ($\mathrm{F}_{\mathrm{init}}$). Such a master key is broken into two parts, used for long term encryption and long term authentication, respectively. A UC-secure authenticated channel is built up in $\mathrm{F}_{\mathrm{init}}$–hybrid model against static adversaries, where shared MAC uses long term shared key and therefore different instances access this primitive by JUC-technique (see e.g. Walfish [24] Th.3.5). Next, smt-channel is

realized in F_{auth}-hybrid model against a static adversary using ind-CPA encryption and JUC approach.

Note, smt-channel could be UC-realized even against full adaptive adversary in F_{auth}-hybrid model using non-committing public key encryption (e.g. in case, when authenticated channels were guaranteed physically), however, here we wanted uniform assumptions against the adversary along the chain of hybrids as well as a symmetric key approach.

UC-secure key exchange is trivial over UC-secure SMT channel. Note, the encrypt-then-MAC paradigm is also followed.
□

Example 14 (*Otway-Rees protocol*)**:**
Consider the following (modified) Otway-Rees (O-R) session key-transfer protocol

1. A→B: $A|\ N_A$ (1)
2. B→S: $A|\ B|\ N_A|\ N_B$
3. S→B: $\{N_A|\ A|\ B|\ k\}K_{AS}$, $\{N_B|\ A|\ B|\ k\}K_{BS}$
4. B→A: $\{N_A|\ A|\ B|\ k\}K_{AS}$

Shared long term keys are pre-assigned. In the UC-approach, environment Z chooses a *sid* and gives it to all parties of the instance. In contrary, in realizations, the *sid* is built up during the run of the instance in several steps; typically, initialized by the party, which sends the first protocol message, then all the others get to know it and contribute to the build-up of the *sid*, when they receive their first message. In O-R protocol

$sid=(A,B, N_A,N_B)$,

where A and B are party identifiers, N_A and N_B are nonces, sent by A and B, respectively. The intention of the designer is to realize smt-channel. Assume, for a moment, already there exist ideal smt-channels in both direction. Over an F_{smt}-hybrid, the protocol simplifies to:

1. S→A: k (2)
2. S→B: k

Protocol (2) is UC-secure (over smt-channels) (the simulation is trivial). The real question is the security of the O-R realization of F_{smt}. In protocol (1) the encryption operations are "overloaded": using a single encryption and the redundancy of the message, the designer intended to realize both an authentic and a private channel.

Consider first the task of authentication, which, in turn, consists of two subtasks; party identification and integrity protection. The integrity of the message is checked first, and if this is successful then the step of identification follows.

Let $I \subset V_2^n$ denote the set of identifiers, which are n-bit strings. Let's assume, when a fake encrypted message is produced without knowing the key, then ideally, it decodes into a random message. Under this assumption the probability of event of "false accept of integrity" becomes:

P(false accept of integrity) $= \left(|I|-1\right)/2^{2n}$.

For instance, for a system with 2^8 parties and 32 bit long identifiers this probability is $\sim 2^{-56}$. Note, we need non-malleable encryption (e.g. CCA2 secure encryption) to prevent the modification of the identifier behind encryption.

Returning to the provable security in the UC-framework, the main problem here is with the double exploitation of a single encryption. The tasks of authentic channel and private channel are independent and formal security proof assumes independent keys, furthermore the design has to follow the encrypt-then-MAC paradigm.
□

Example 15 (*Needham-Schroeder-Lowe protocol*)**:**
The Needham-Schroeder-Lowe session key protocol was analyzed in [3] using BPW's approach together with the composable cryptolibrary [4]. This is a timed protocol and time is not modeled in cryptolibrary [4], therefore the authors of [3] substituted time with nonces (cryptolibrary [4] includes nonces). After such simplification, the protocol was proved to be secure in the BPW's framework.
In this public key protocol the *sign-then-encrypt* technique [2] securely realizes the sequential composition of the corresponding authentic and private channels, relying on PKI setup providing public key certificates.
Security proved in [3] is essentially equivalent to (base) UC-security. In Chapter 4, we will return more detailed to the corresponding aspects of the BPW's approach as well as to the problem of time modeling.
□

4. Symbolic analysis and composable cryptolibraries

Up to this chapter we considered the composable design and analysis in the (Canetti's) UC-framework, where the cryptographic task was decomposed into subtasks and the proof went by subtasks in hybrid models. Here we go down to more granular cryptographic building elements, to the level of cryptographic primitives. In this chapter we consider the corresponding BPW's approach and provide comments to its relationship with the UC-approach.

4.1. Analysis in the BPW's framework

When we analyze a protocol by the BPW's approach, the real cryptographic primitives are substituted by ideal primitives using the composable cryptolibrary. If the protocol contains only library primitives, then such a substitution results in a deterministic, symbolic protocol, where cryptography (with randomness and computational complexity assumptions) is completely eliminated. The next step is to show that for honest parties this deterministic protocol provides the security service according to the security goal against. The elimination of the cryptographic details largely simplifies the proof of simulatability at the service interface and opens the potential for an automation of the analysis. Using the terms of Section 3, by the elimination of the real primitives we arrive to a hybrid protocol.

Note, the output at the service interface, which is checked by the analysis, is only a part of the global output. One might miss mentioning the interface between the environment and the adversary in this sketchy description of the analysis: black box simulation is carried out, where we have to define a simulator, which "translates" between the ideal primitives and the real adversary in an indistinguishable way for the real adversary.

The technique of invariants is a valuable tool in proving the security requirements on symbolic protocols. Invariants of the symbolic system are statements about the state of the symbolic system which hold at all times in all runs of the system. When proving the invariant, we prove, that if an invariant holds at time t in a run of the system it will still hold at time $t+1$ (one time unit is needed for the system to make one step). For the application of the BPW's framework in the analysis, we mention [18], [19] for secure routing protocols, [22] for secure anonymity protocols. Non-malleable public key encryption within BPW's framework is discussed in [21]. In [20] this terminology is used for the definition and analysis of an UC-secure hash function.

4.2. The Canetti's and the BPW's approach

BPW's approach is essentially equivalent to a base UC approach of Canetti. In the BPW's approach, different instances of the same or different protocols use common primitives. There is a common public key encryption/decryption primitive, a common digital signature primitive etc.

The ideal model of a primitive differs from the Canetti's style (UC) of ideal functionality, which describes the functionality of a single instance. At the level of the main protocol, which uses different primitives once or several times during an invocation, all the primitives are turned into their ideal model resulting in a multi-hybrid protocol. The main entity in the ideal system is called trusted host (TH). TH starts running with the first invocation of an instance of the protocol. TH runs the multi-hybrid protocol; receives commands to different primitives and "forwards" them to the ideal model of them; stores all the usages during the lifetime of the protocol together with all the corresponding items.

In the BPW's ideal model of a primitive the *whole history* of its usage is stored. In the UC approach, session identifiers separate the instances. In the BPW's approach the access to the cryptographic objects (produced during the run and stored in the trusted host (plaintexts, keys etc.)) are controlled by so called handles. Those participants, which have a *handle* (to a stored entry corresponding) to a secret key, have access to the given key and are able to do decryption. If an adversary corrupts a party then it inherits all the handles of the party.

Figure 5 illustrates the access to a common ideal functionality of a cryptographic primitive (ρ) by two different instances (π_1 and π_2) of the main protocol π. The common functionality ρ stores the history of all of its invocations. Figure 6 illustrates the corresponding UC (JUC) approaches. Each main instance (π_1 as well as π_2) invokes its own, independent instance of ρ in case of (base) UC. JUC arrangement shows some resemblance to Figure 5 by the common access to an extended ideal functionality $\overline{\rho}$, however, here sub-sid's separate the different invocations and within $\overline{\rho}$ separate ρ-instances are "running". This hierarchical structure of sid's in the UC-approach cannot appear in the BPW's approach, where handles are "PID-associated" and not "instance-associated".

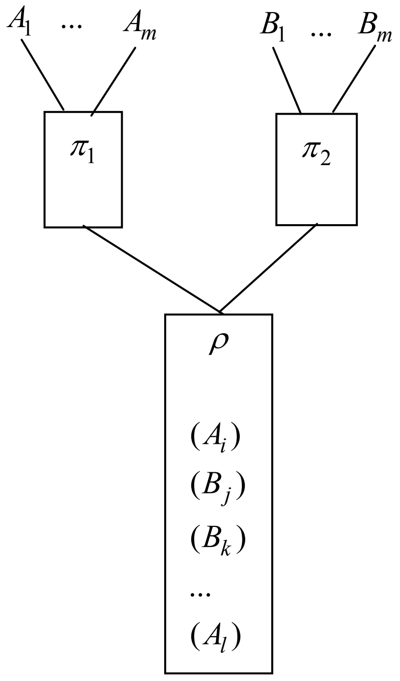

Fig.5: Trusted host in BPW's approach

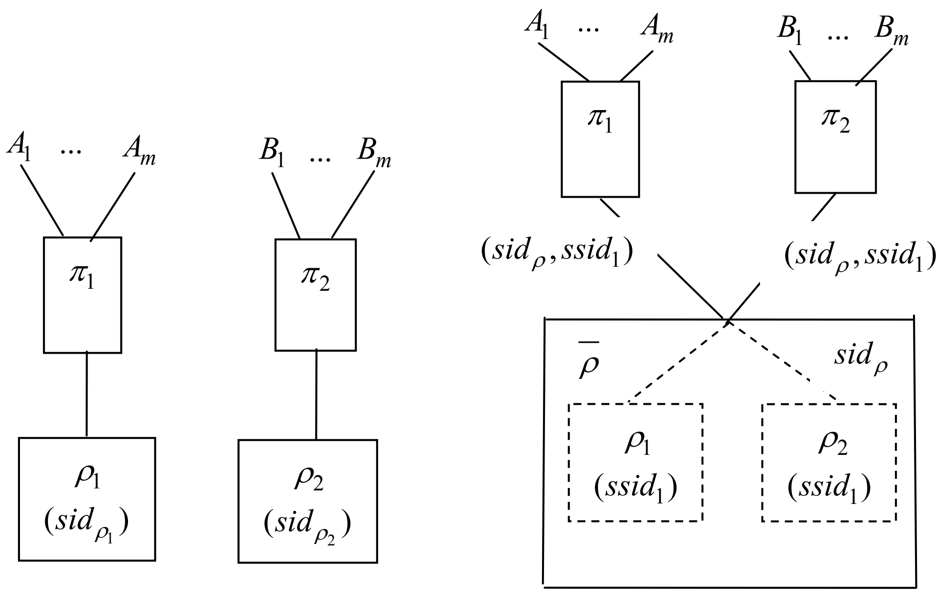

Fig.6: UC- and JUC-approaches (left and right, resp.)

From an application point of view, the main conceptual difference between the two approaches is the following:

- in UC the aim is to reduce the task of the definition/design/analysis down to a single instance to make the approach (the proof of simulatability) as simple as possible;
- in the BPW's approach the aim was to transform the protocol (safely, i.e. in a cryptographically sound way) into a version which is amenable even for an automated analysis by eliminating all computationally secure elements (cryptographic primitives) and arriving to a purely deterministic (hybrid) protocol.

Nevertheless, it seems that the BPW's trusted host ("all-in-one") model carries the possibility to bring it closer to the UC's single instance approach:

Let's consider a target instance and extract (by appropriate indexing) all its references from the database of the TH and, by this way, separate the database entries of the target instance from all the other (past or concurrent) instances (e.g. in Figure 5, entries corresponding to PIDs $A_j ... A_l$). The access control approach (implemented by handles) in the BPW's ideal models simply define the expected secure separation of "items" coming from the target instance and all other instances, as well as separation of "items" within the target instance. Informally, by this way, we avoid unwanted interaction between different instances potentially induced by the adversary. This observation is formed as:

The BPW's trusted host model involves the single instance approach by appropriately sorting the entries stored in the trusted host.

BPW's approach provides an essentially equivalent UC assessment. From an application point of view, it arises also a corresponding question: How much we loose, if we analyze "just" at (base) UC / BPW level instead of GUC?

Assume we have a shared functionality used by our protocol, but instead of GUC we do the analysis in UC framework. Recall, the special advantage of the GUC approach comes ahead in case of full adaptive adversary. In case of static adversary there is no difference between the usefulness of the two approaches.

4.3. Analysis of timed protocols: "random-time" server

Example 15 mentions the analysis of modified (weaker) Needham-Schroeder-Lowe session key protocol in BPW's approach, where time was substituted with nonces.

Note, time can be considered as a special shared functionality T in a GUC approach. The i-th sample is not taken (independently) from a distribution (like in case of CRS), but from a space $[t_i, \infty)$ of real numbers, where $t_1 < t_2 < ... < t_i$. If within an instance, there are m time elements $t_{(1)} < t_{(2)} < ... < t_{(m)}$ then the *sid* of this instance is (essentially) $[t_{(1)}, t_{(2)}, ..., t_{(m)}]$.

Those protocol messages, which carry also time elements are (expectedly) secured by authentication/encryption. If such a message is involved in commitment during the simulation (of honest messages), then it is necessary to place the time elements correctly into the message to escape a simulation failure during corruption. This can be done for instance in case of public key encryption, but surely not in cases, when the key stuff is not available for the simulator (e.g. symmetric key primitives; encryption, MAC or digital signature). Note, however, there is a more fundamental problem with (real) time as a shared functionality: the ideal system should run in time synchrony with the real system. Indeed, environment Z as a distinguisher, which has access to the shared functionality of time, should not be able to distinguish timing incoherence between the run of the two systems.

Therefore we give up some of our ambition (with global time) and we try to remain within the UC framework. We reinforce this similarity: assume a *"random-time" server (RTS)*, which on a request replies with a "random time", which is accessible only for the parties of the given instance. The timed protocol modified (weakened) by relying on RTS setup instead of real time functionality T, similar to substitution by nonces ([3]) referred in Example 15, however, here we placed the problem in a somewhat wider picture.

Summarizing the above ideas, we conjecture that:

Conjecture 2:
(a) Timed protocols cannot be analyzed in the natural model of global time as a shared functionality.
(b) Within the UC framework, the natural way of analysis is in the "random-time" server (RTS) setup.

Note, only a weakened version of the protocol can be analyzed this way. We drop essential properties of time and loose valuable information: the comparison of different samples (earlier/later) and the (time-) distance between different timed events.

The roots of the problem are deeper then just an "UC or GUC" debate:
(c) The problem with functionality T is a special case of dependent sampling from a distribution (i.e. instead of independent sampling).

Note, RTS shows some resemblance to the CRS (Common Random String) setup functionality. The point here is to examine, if by this way we could inherit also results corresponding to CRS setup functionality in the UC framework.

In [23] we proposed an alternative way for the analysis of time-aware protocols. In particular, we introduced an *event-driven clock* (e-time) and discussed a few properties of e-time relevant to an analysis.

Conjecture 3: *The event-driven clock (e-time) approach is the natural way to catch the closest notion to functionality T in the emulation of an ideal process approach of security for time-aware protocols.*

References

[1] M. Abadi and R. Needham. Prudent Engineering Practice for Cryptographic Protocols. IEEE Transactions on Software Engineering, January 1996.

[2] J.H. An, Y. Dodis and T.Rabin. On the Security of Joint Signature and Encryption. Advances in Cryptology. EUROCRYPT 2002 (L.R.Knudsen ed.) , LNCS, Vol. 2332, Springer, 2002, pp. 83-107.

[3] 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.*

[4] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. *IACR Cryptology ePrint Archive*, Report 2003/015, http://eprint.iacr.org/, January 2003.

[5] 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.

[6] M. Backes, B. Pfitzmann, and M. Waidner. Limits of the Reactive Simulatability/UC of Dolev-Yao Models for Hashes. *Cryptology ePrint Archive: Report 2006/068.* also in *Workshop on Formal and Computational Cryptography (FCC 2006) (2006)*

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

[8] 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, 2005, *Proceedings* CRYPTO 2005.

[9] R.Canetti, U. Feige, O. Goldreich and M. Naor: Adaptively Secure Multi-party Computation or Non-Commiting Encryption. Adaptively secure multi-party computation. *In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pp. 639-648, Philadelphia, Pennsylvania, May 1996.

[10] R. Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptolology*. Vol. 13, No.1, 2000

[11] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. *Cryptology ePrint Archive: Report 2000/067.* (received 22 Dec 2000, revised 13 Dec 2005)

[12] 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.

[13] R.Canetti and T.Rabin. Universal Composition with Joint State. *Crypto'03*, 2003.

[14] R.Canetti, Y.Dodis, R.Pass and S.Walfish. Universally Composable Security with Global Setup. Cryptology ePrint Archive: Report 2006/432. 20 Nov 2006

[15] A. Hevia, D. Micciancio. An indistinguishability-based characterization of anonymous channels. In Nikita Borisov and Ian Goldberg, editors, *Privacy Enhancing Technologies: Eighth International Symposium, PETS 2008*, pages 24-43. Springer-Verlag, LNCS 5134, July 2008.

[16] H. Krawczyk. The order of encryption and authentication for protecting communications (Or: how secure is SSL?). http://eprint.iacr.org/2001.

[17] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM CCS*, pages 245–254, 2000

[18] I.Vajda. Cryptographically Sound Security Proof for On-Demand Source Routing Protocol EndairA. *Cryptology ePrint Archive Report 2011/103*. http://eprint.iacr.org/2011/103.pdf

[19] I.Vajda. Framework for Security Proofs for Reactive Routing Protocols in Multi-Hop Wireless Networks. *Cryptology ePrint Archive Report 2011/237*. http://eprint.iacr.org/2011/237.pdf

[20] I.Vajda. New look at impossibility result on Dolev-Yao models with hashes. *Cryptology ePrint Archive Report 2011/335*. http://eprint.iacr.org/2011/335.pdf

[21] I.Vajda. Non-malleable public key encryption in BRSIM/UC.
Cryptology ePrint Archive Report 2011/470. http://eprint.iacr.org/2011/470.pdf

[22] I.Vajda. UC framework for anonymous communication.
Cryptology ePrint Archive Report 2011/682. http://eprint.iacr.org/2011/682.pdf

[23] I.Vajda. On instance separation in the UC-framework. Cryptology ePrint Archive: Report 2012/302. 30 May, 2012

[24] S.Walfish. Enhanced Security Models for Network Protocols. PhD Dissertation. New York University. January 2008.