Capítulo

4

Introdução à Composibilidade Universal

João J. C. Gondim
Departamento de Ciência da Computação
Universidade de Brasília

Resumo

A composibilidade universal - universal composability (UC) é um framework geral que se propõe representar protocolos criptográficos e analisar sua segurança. Neste framework, a segurança dos protocolos é preservada sob uma operação geral de composição com outros protocolos, permitindo o design modular e a análise de protocolos complexos a partir de blocos mais simples de forma unificada e sistemática. Além disso, neste framework a segurança dos protocolos é mantida sob uma operação geral de composição. Esta operação é definida de tal forma que, satisfazendo as condições desta operação, é possível construir protocolos que não só são seguros mas também tem sua segurança preservada quando compostos entre si, mesmo na presença de um número qualquer de cópias concorrentes. A proposta é apresentar no minicurso os conceitos básicos da segurança UC e sua aplicação na concepção e análise de um protocolo criptográfico. Apesar de já fazer dez anos da apresentação do conceito, contando com vários resultados relevantes, ele ainda é pouco conhecido.

Inicialmente, é feita toda a modelagem de como vem a ser o modelo de execução dos protocolos. Apesar da motivação inicial ter sido protocolos relacionados a primitivas criptográficas, a abordagem se aplica a protocolos de computação segura multipartes, como indicado em trabalhos recentes. O modelo de computação será construído a partir da noção de sistemas de máquinas de Turing interativas. Na sequência é apresentado o framework, começando com a definição de dois modelos específicos que diferem no modelo de comunicação entre as partes e do adversário. Pode-se então apresentar o Teorema da Composição. Com o Teorema da Composição, duas aplicações são abordadas,

no contexto dos protocolos de comprometimento de bit. Primeiro é mostrado um exemplo de como o framework pode ser usado para analisar os requisitos para a construção segura de protocolos de comprometimento de bit. Especificamente, é demonstrado que não há implementação segura sem que se recorra a hipótese de setup. Na sequência, se constrói um protocolo de comprometimento de bit e se estabelece a segurança UC do mesmo.

4.1. Introdução

A demonstração rigorosa da segurança de um protocolo é um requisito indispensável no projeto de um protocolo criptográfico. Isto requer pelo menos dois passos de formalização. O primeiro envolve a definição do modelo matemático apropriado para representar o protocolo, enquanto o outro consiste em formular nesse modelo uma definição de segurança que capture os requisitos para a tarefa que o protocolo pretende realizar seguramente. Com a definição de segurança, pode-se então estabelecer que o protocolo é seguro demonstrando que sua representação matemática satisfaz a definição de segurança no modelo adotado.

Entretanto, a formulação de um modelo matemático apropriado para representação dos protocolos e a formulação de uma definição de segurança adequada no modelo escolhido são tarefas complexas. O modelo deve ser capaz de capaz de representar todos os comportamentos adversariais realistas e a definição deve garantir que a noção intuitiva de segurança foi capturada corretamente com respeito a qualquer comportamento adversarial que seja considerado.

Outro aspecto importante no contexto da segurança de um protocolo criptográfico é a sua robustez com relação ao ambiente de execução. O comportamento de um protocolo criptográfico sofre grande influência de seu ambiente de execução, em particular com relação aos outros protocolos que estejam executando no mesmo sistema ou rede. Consequentemente, um critério que precisa ser incorporado à definição de segurança é a garantia de robustez ao ambiente de execução, que pode ser colocada de forma mais geral como uma forma de modularidade. Isto é, a necessidade de se garantir a segurança quando o protocolo é usado como um componente dentro de um sistema maior.

Tradicionalmente, primitivas criptográficas são desenvolvidas inicialmente como tarefas isoladas, sem levar em consideração ambientes de execução mais complexos. esta abordagem tem vantagens como permitir definições mais concisas e intuitivas além de simplificar a análise dos protocolos. Entretanto, em muitos casos estas definições iniciais se mostram insuficientes quando as condições do ambiente são refinadas aumentando em complexidade onde os protocolos são empregados em situações mais gerais. Alguns exemplos desse tipo de situação que podem ser citados, entre outros, são:

- cifração: a noção básica de segurança semântica [Goldwasser and Micali 1984] foi depois refinada de várias formas de segurança ataques CCA (como [Naor and Yung 1990], [Dolev et al. 2000], [Rackoff and Simon 1992], [Bellare et al. 1998b]) e segurança adaptativa (como por exemplo [Beaver and Haber 1992], [Canetti et al. 1996]), visando satisfazer situações mais gerais;

- comprometimento: onde as noções iniciais foram estendidas com conceitos como não maleabilidade ([Dolev et al. 2000], [Di Crescenzo et al. 1998], [Fischlin and Fischlin 2000]) e equivocação ([Brassard et al. 1988], [Beaver 1996]);

- zero knowledge: onde se demonstrou que as noções iniciais ([Goldwasser et al. 1989], [Goldreich and Oren 1994]) não eram fechadas sob composição paralela e concorrente, levando a necessidade de novos conceitos e construção ([Goldreich and Krawczyk 1990], [Dwork et al. 1998], [Canetti et al. 1999], [Barak et al. 2001]);

- troca de chaves: onde o conceito original não era suficiente para garantir sessões seguras (como em [Bellare and Rogaway 1994], [Bellare et al. 1998a], [Shoup 1999], [Canetti and Krawczyk 2001]);

- oblivious transfer: como em [Rabin 1981], [Even et al. 1985], [Garay and MacKenzie 2000].

Uma forma de capturar as questões de segurança que surgem em ambientes de execução específico ou em uma dada aplicação é representar o ambiente ou a aplicação diretamente dentro de uma extensão da definição de segurança. Essa abordagem é a adotada, por exemplo, em casos de troca de chaves [Bellare and Rogaway 1994], [Canetti and Krawczyk 2001], comprometimento não maleável [Dolev et al. 2000], e em zero-knowledge concorrente [Dwork et al. 1998], onde a definição modela explicitamente várias instâncias do protocolo em questão que são coordenadas pelo adversário. Essa abordagem tem a desvantagem de resultar em definições cada vez mais complexas e é inerentemente limitado em escopo pois trata apenas de ambientes e questões específicas.

Uma abordagem alternativa, adotada no *framework* UC é tratar os protocolos como *stand alone* mas garantir que sua composição é segura. Isto é, as definições de segurança se aplicam na inspeção de uma instância isolada do protocolo. Em situações complexas, onde uma instância de um protocolo pode ser executada concorrentemente com outras instâncias, com entradas arbitrárias e possivelmente com controle do adversário, a segurança é garantida se preservada sob uma operação geral de composição de protocolos. Esta abordagem simplifica consideravelmente o processo da formulação de uma definição de segurança e da análise do protocolo. Além disso, a segurança do protocolo é garantida em ambientes arbitrários, mesmo os que não tenham sido explicitamente considerados.

A abordagem descrita acima, juntamente com sua operação de composição, tem como requisito básico a formulação de um *framework* geral em que se possa representar os protocolos criptográficos e seus requisitos de segurança. Várias definições gerais de protocolos seguros foram propostas, como por exemplo [Goldwasser and Levin 1991], [Micali and Rogaway 1992], [Beaver 1991], [Ben-Or et al. 1993], [Pfitzmann and Waidner 1994], [Canetti 1998], [Hirt and Maurer 2000], [Pfitzmann et al. 2000], [Dodis and Micali 2000], [Pfitzmann and Waidner 2000], sendo candidatas para esse *framework* geral. Entretanto, além de restrições de escopo, seja nas situações ou tarefas às quais se aplicam, elas

não oferecem garantia geral de segurança na composição de protocolos criptográficos. Isto é o caso especialmente em situações em que os adversários são computacionalmente limitados e várias instâncias do protocolo executam simultaneamente com possível controle do adversário.

As motivações listadas acima formam o princípios que levaram ao desenvolvimento do *framework* UC, voltado para a representação e análise de protocolos criptográficos. O *framework* UC define uma metodologia geral para expressar os requisitos de segurança de protocolos criptográficos. Além disso, provê um método geral de composição de protocolos, de forma que a definição de segurança expressa no framework se preserva sob a composição, chamada de *composição universal*.

De forma extremamente resumida, a composição universal pode ser vista de maneira intuitiva como uma generalização da operação de chamada de subrotina em algoritmos sequenciais para ambientes distribuídos com execuções concorrentes de protocolos. A preservação da segurança no *framework* implica que um protocolo seguro permanecerá seguro mesmo quando executando em um ambiente arbitrário e desconhecido multiparte multiexecução.

Recentemente, o *framework* UC tem sido aplicado para estabelecer a segurança composicional de aplicações com funcionalidades mais gerais que protocolos criptográficos. Como exemplo, pode-se citar dentre outras aplicações: serviços de sistemas operacionais [Canetti et al. 2010], produto interno [Dowsley et al. 2010], autenticação [Chari et al. 2011], troca de chaves [Canetti and Gajek 2010] [Canetti et al. 2005] [Gorantla et al. 2009], incoercibilidade [Unruh and Müller-Quade 2009], criptografia simétrica [Kuesters and Tuengerthal 2009], certificação digital [Canetti 2003], assinatura digital [Kurosawa and Furukawa 2008], análise simbólica de protocolos criptográficos [Canetti and Herzog 2004], computação síncrona [Katz et al. 2011], compartilhamento de segredo [Dowsley et al. 2009], análise de protocolos [Green and Hohenberger 2008], [Gajek et al. 2008], entre outras aplicações. Isto reforça a formulação geral do *framework* UC, deixando claro sua aplicabilidade ao contexto mais geral da computação segura multipartes.

4.1.1. Breve descrição deste capítulo

Este capítulo está organizado em seções. Na que segue, é construído o modelo computacional sobre o protocolo e o ambiente, juntamente com o adversário, executam. Depois a *framework* UC é apresentada com os modelos adversariais e na seção seguinte é enunciado o Teorema da Composição. As duas seções seguintes ilustram aplicações do *framework* UC tomando como base protocolos de comprometimento de bit.

Primeiro, é apresentado e demonstrado como o *framework* pode ser utilizado na análise de um protocolo, identificando condições para sua implementação segura. Depois, um protocolo de comprometimento de bit UC seguro é construído, seguindo as recomendações da análise feita anteriormente.

O capítulo é encerrado com conclusões e comentários finais.

4.2. O modelo básico de computação

Antes de iniciar a apresentação do framework UC, o modelo computacional é descrito. A formalização de algoritmos em uma rede de comunicação como máquinas de Turing interativas, ITM, segue a abordagem de [Goldwasser et al. 1989]. Uma descrição detalhada, voltada à formalização das interações entre pares de máquinas pode ser encontrada em [Goldreich 2000].

Uma fita de uma máquina de Turing é dita ser write-once, **wo**, se sua cabeça de leitura se move em apenas uma direção. Se uma fita pode se escrita por outras máquinas de Turing, diz-se que ela é externamente write-once, **ewo**. Se a fita pode ser lida e escrita, diz-se que ela é **rw**

Definição 4.2.1. *Uma máquina de Turing interativa (ITM) M tem as seguintes fitas:*

- *uma fita* **ewo** *de identidade*
- *uma fita* **ewo** *de parâmetro de segurança*
- *uma fita* **ewo** *de entrada*
- *uma fita* **ewo** *de comunicação*
- *uma fita* **ewo** *de saída de subrotina*
- *uma fita de saída*
- *uma fita aleatória*
- *uma fita* **rw** *de ativação (1 bit)*
- *uma fita* **rw** *de trabalho*

O conteúdo da fita de identidade é chamado de identidade de M. A identidade de M é interpretada, segundo alguma codificação, como dois strings: o identificador de sessão SID de M, e o identificador da parte PID de M.

O conteúdo da fita de comunicação modela a informação vinda da rede. Ela é interpretada, segundo alguma codificação, como uma sequência de valores chamados mensagens, onde cada mensagem tem dois campos: o campo com a designação do emissor, que corresponde à identidade de alguma ITM, seguido por um campo arbitrário de conteúdo.

O conteúdo da fita de saída de subrotina modela as saídas das subrotinas de M. Ela é interpretada, segundo alguma codificação, como uma sequência de valores chamados saídas de subrotinas, onde cada uma tem dois campos: o campo com o identificador de subrotina, que corresponde à identidade de alguma ITM junto com o código de alguma ITM, e por um campo arbitrário de conteúdo.

O código de alguma ITM M pode incluir instruções para escreve na fita de outra ITM. A sintaxe e o efeito destas instruções são descritos mais adiante.

Definição 4.2.2. *Um sistema de ITMs $S = (I, C)$ é definido por uma ITM inicial I e uma função de controle $C : \{0,1\}^* \rightarrow \{allow, disallow\}$ que determina o efeito das instruções escritas externamente nas ITMs do sistema.*

Definição 4.2.3. *Uma configuração de uma ITM M é composta por:*

- *código;*
- *estado de controle;*
- *conteúdos de todas as fitas;*
- *posições de todas as cabeças*

Uma configuração está ativa se o bit da fita de ativação tem valor 1; caso contrário está desativada.

Definição 4.2.4. *Uma instância $\mu = \{M, id\}$ de uma ITM, ITI, consiste do código M juntamente com o string de identidade $id \in \{0,1\}^*$*

Diz-se que uma configuração é uma configuração de uma instância μ se o código da configuração é M e o conteúdo da fita de identidade é id.

Definição 4.2.5. *Uma ativação de uma ITI μ é uma sequência de configurações que correspondem a uma computação, que inicia a partir de alguma configuração ativa de μ, até que uma configuração inativa seja alcançada. Nesse caso, diz-se que a ativação se completou e que μ está esperando a próxima ativação.*

Se um estado especial de parada, **`halt`**, for atingido a ITI para; nesse caso, a ITI não executará nada em ativações futuras.

Definição 4.2.6. *Uma execução de um sistema $S = (I, C)$, dado um parâmetro de segurança k e entrada x, é uma sequência de ativações de ITIs. A primeira ativação inicia a partir da configuração com o código de I, a entrada x na fita de entrada, 1^k escrito na fita do parâmetro de segurança, um string aleatório r, longo o suficiente, escrito na fita aleatória, e identidade 0.*

A ITI $(I, 0)$ é chamada de ITI inicial. A execução termina quando a ITI inicial atinge o estado **`halt`**, i.e. quando uma configuração de parada da ITI inicial é atingida. A saída gerada pela execução é o conteúdo da fita de saída da ITI inicial na sua configuração de parada.

Para completar a definição de uma execução ainda é necessário descrever o efeito de uma instrução de escrita externa; e como se determina qual a próxima ITI a ser ativada, após se completar uma ativação.

4.2.1. Escrevendo na fita de outra ITI e chamadas de ITIs

Uma instrução de escrita externa por uma ITM em outra, especifica os seguintes parâmetros: os códigos e identidades das ITIs de origem e destino, a fita do destino que será escrita, os dados que serão escritos. A fita de destino pode ser a fita de entrada ou a fita de comunicação, ou a fita de saída de subrotina. O efeito de uma instrução de escrita externa de uma ITI $\mu = (M, id)$ para uma ITI $\mu' = (M', id')$ é definido como segue:

1. se a função de controle C aplicada à sequência de pedidos de escrita externa até o momento não permite que μ escreva na fita especificada de μ', i.e. retorna um valor *disallow*, então a instrução é ignorada.

2. se C permite a operação e uma ITI $\mu'' = (M'', id'')$ com identidade $id' = id''$ já existe no sistema (uma das configurações na execução até agora tem uma identidade id'), então:

   (a) se a fita do destino é a fita de comunicação de μ', então os dados especificados são escritos na fita de comunicação de μ'', juntamente com a identidade *id* da ITI de origem. Consequentemente, uma nova configuração μ'' é gerada. Essa configuração é a última configuração de μ'' nesta execução, com a nova configuração sendo escrita na fita de comunicação. Isto acontece independente do código M'' de μ'' ser igual ao código M' especificado no pedido de escrita externa.

   Esta regra assume que uma ITI não conhece necessariamente o código da ITI para a qual manda ou da qual recebe mensagens.

   (b) se a fita do destino é a fita de saída de subrotina de μ', então os dados especificados são escritos na fita de saída de subrotina de μ'' juntamente com o código e a identidade de μ. Novamente, isso acontece independente de $M'' = M'$.

   Esta regra assume que uma ITI conhece o código da ITI que escreve para sua fita de saída de subrotina. Entretanto, uma ITI não conhece necessariamente o código da ITI para a qual ela gera uma saída.

   (c) se a fita do destino é a fita de entrada de μ' e $M'' = M'$, então os dados especificados são escritos na fita de saída de subrotina de μ'' juntamente com o código e a identidade de μ. Se $M' \neq M''$ então μ vai para um estado especial de erro.

   Esta regra assume que uma ITI pode verificar o código da ITI para a qual provê entradas. Entretanto, uma ITI não conhece necessariamente o código da ITI para da qual ela recebe entradas.

3. se C permite a operação, e nenhuma ITI com identidade id' existe no sistema, então uma nova ITI com código M' e identidade id' é chamada, i.e. uma nova configuração é gerada, com código M' e identidade id' na fita de identidade, 1^k escrito na fita do parâmetro de segurança, um *string* aleatório r, longo o suficiente, escrito na fita aleatória. Uma vez que a nova ITI é chamada, a instrução de escrita externa é executada como acima. Nesse caso diz-se que μ chamou μ'.

Note que as regras de chamada das ITIs, juntamente com o fato da execução começar com uma única ITI, garante que cada ITI no sistema tenha identidade única. Assim, quaisquer duas ITI não tem mesma identidade, independente de seus códigos. Além disso, uma vez que o código e a identidade da ITI de destino devem ser especificados na instrução de escrita externa, estes devem ser computados pela ITI de origem antes da chamada. Assim, não há restrições para SID e PID.

4.2.2. Determinação da ordem de ativações

A ordem de ativações é bem simples. A cada ativação permite-se que cada ITI execute no máximo uma instrução de escrita externa. A ITI cuja fita foi escrita durante uma ativação será a próxima a ser ativada, i.e. a fita de ativação nesta nova configuração da ITI tem o valor 1. Se nenhuma instrução de escrita externa foi executada, a ITI inicial será ativada na sequência.

Esta regra é simples e natural, permitindo quebrar a computação distribuída em uma sequência de eventos locais onde a cada instante de tempo tem-se apenas uma ITI cujo estado local mudou desde sua última ativação.

Quando uma ITI μ escreve uma mensagem m na fita de comunicação de uma ITI μ', diz-se que μ enviou m a μ'. Quando uma ITI μ escreve um valor x na fita de entrada de uma ITI μ', diz-se que μ passou x a μ'. Quando uma ITI μ' escreve um valor x na fita de saída de subrotina de uma ITI μ, diz-se que μ' passou a saída (ou apenas passou) x para μ. Diz-se que μ' é uma subrotina de μ, se μ passou uma entrada para μ' ou se μ' passou uma saída para μ. μ' é uma subsidiária de μ, se μ' é uma subrotina de μ ou de alguma de suas subsidiárias.

4.2.3. Protocolos

Um protocolo é definido como uma ITM, representando o código a ser executado por cada participante.

Definição 4.2.7. *Dado um estado de um sistema de ITMs, a instância do protocolo multipartes π com SID sid é o conjunto de ITMs no sistema cujo código é π e cujo SID é sid.*

Assim, os PIDs nas instâncias dos protocolos são necessariamente diferentes. Assume-se que π ignore todas as mensagens onde o SID é diferente do SID local. Cada ITI em uma instância de um protocolo é chamada de parte. A definição de estado de um sistema de ITIs é apresentada mais adiante.

Definição 4.2.8. *Uma subparte é uma subrotina de uma parte ou de outra subparte.*

Definição 4.2.9. *Uma instância estendida de π inclui todas as partes e subpartes desta instância.*

Deve-se notar que na definição acima é feita uma distinção entre o protocolo, que representa o código que será executado por cada parte e instância do protocolo que representa uma execução específica de um protocolo. Além disso, a associação de um SID

com cada instância do protocolo, onde SID é conhecido por todas as partes, é conveniente para modelar múltiplas instâncias do protocolo executadas concorrentemente no mesmo sistema.

Definição 4.2.10. *O estado de um sistema de ITMs representa uma descrição completa de um certo instante da execução do sistema. Especificamente, consiste da sequência de todas as configurações de todas as ativações do sistema até o dado instante.*

Definição 4.2.11. *O registro de uma execução de um sistema é o estado final de execução, onde a última configuração é terminal para a ITI inicial.*

Definição 4.2.12. *Um sistema estendido é um sistema em que a função de controle pode alterar instruções de escrita externa. Em um sistema $S = (I,C)$, a função de controle C toma como entrada uma sequência de instruções de escrita externa e gera como saída valores como allowed ou disallowed. Em um sistema estendido, a saída de C consiste em uma instrução de escrita externa completa, que pode ser diferente do pedido original de escrita externa. A instrução executada é a saída de C.*

Essa capacidade da função de controle poder alterar instruções de escrita externa será utilizada para modificar o código de ITIs geradas durante a execução.

4.2.4. Sistemas e ITMs probabilísticas e em tempo polinomial

Apesar de não ter sido dito até então, as ITMs de interesse são limitadas em recurso. Especificamente, o foco da modelagem recai sobre as ITMs que operam em tempo polinomial. Entretanto a definição rigorosa da computação com recursos limitados em um ambiente interativo requer certos cuidados. Isto se aplica ao caso em questão em que o cenário é dinâmico, com ITMs sendo geradas durante a evolução do sistema.

O ponto de partida é considerar uma ITM *M* como PPT, probabilística em tempo polinomial, se seu tempo total de execução, em todas as suas ativações, é polinomial no comprimento da sua entrada. Entretanto, esta definição estaria mais apropriada para capturar situações envolvendo pares de ITMs cujas entradas são fixas durante a computação. Assim ela será estendida a cenários mais gerais, onde as instâncias de ITMs podem escrever arbitrariamente nas fitas de outras instâncias ao longo da execução. Especificamente, será necessário limitar o tempo total de execução de um sistema de ITMs. Em particular, o tempo total de execução de cada instância e o número de instâncias deverá ser limitado.

Assim, para garantir esse limite geral, além da condição do tempo ser polinomial sobre o tamanho da entrada serão levados em consideração o parâmetro de segurança, junto com o número de bits escrito em outras ITMs e o número de ITMs que foram escritas. Desse total deve-se descontar o número de bits escritos na ITM.

Desta forma o *n* para o qual a ITM *M* será polinomial, será a soma do parâmetro de segurança *k*, mais o total de bits escritos até então na fita de entrada de *M*, menos os bits escritos por *M* nas fitas de outras ITMs. Deve-se ainda subtrair *k* vezes o número de máquinas em que *M* escreveu. As subtrações são motivadas por: subtrair os bits escritos por *M* nas fitas de outras ITMs leva em consideração que escrever nas fitas de ITMs existentes não aumenta o tempo de execução; e subtrair *k* vezes o número de máquinas

em que M escreveu leva em consideração que as chamadas a novas instâncias também não aumenta o tempo total de execução.

Além disso, estas condições devem valer durante todo o tempo de execução.

Definição 4.2.13. *Seja $p : N \to N$. Uma ITM M é localmente p-limitada se, a qualquer momento durante sua execução, em qualquer de suas configurações, o número total de passos executados por M é no máximo $p(n)$, onde*

$$n = k + n_I - n_O - k.n_N$$

onde k é o parâmetro de segurança, n_I é o número total de bits escritos na fita de entrada de M, n_O é o número total de bits escritos por M na fita de entrada de outras ITM, e n_N é o número de diferentes instâncias de ITMs cujas fitas foram escritas por M. Se M é localmente p-limitada e, além disso, não faz escritas externas ou cada escrita externa especifica uma ITM que é p-limitada, então se diz que M é p-limitada. M é PPT se existe um polinômio p tal que M é p-limitada. Um protocolo multipartes é PPT se ele é uma ITM PPT.

Pode-se ver que o número total de passos na execução de um sistema, onde a ITM inicial é p-limitada, é limitado por $p(n+k)$, onde n é o tamanho da entrada e k é o parâmetro de segurança. Em particular, o número total de instâncias chamadas é limitado por $p(n+k)$. Se a função de controle C é computável em tempo $q(t)$, onde t é o comprimento da entrada para C, então uma execução de um sistema de ITMs pode ser simulado em uma máquina de Turing sequencial com entrada x e k é o parâmetro de segurança, em $O(q(p(|x|+k)))$ passos, onde $|x|$ é o comprimento de x.

Proposição 4.2.1. *Se a ITM inicial em um sistema (I,C) é p-limitada e a função de controle C é computável em tempo $q(t)$. Então uma execução do sistema pode ser simulada em uma única máquina de Turing não interativa M, com entrada x e parâmetro de segurança k, em $O(q(p(|x|+k)))$ passos (onde $|x|$ é o comprimento de x). Em particular, se I e C são PPT então M também é. Isto também vale para sistemas estendidos de ITMs, se as ITM chamadas forem p-limitadas.*

4.2.5. O modelo de execução de protocolo

O modelo de execução de protocolos é definido em termos de um sistema de ITMs, capturando a noção de uma rede assíncrona, onde as comunicações são não autenticadas e não confiáveis. O modelo em si é parametrizado por três ITMs: o protocolo π a ser executado, o ambiente $\mathscr{E}$ e o adversário $\mathscr{A}$. Assim, dados π, $\mathscr{E}$ e $\mathscr{A}$ o modelo para execução de π é o sistema estendido de ITMs PPT $(\mathscr{E}, C_{EXEC}^{\pi,\mathscr{A}})$, onde a ITM inicial é o ambiente $\mathscr{E}$ e função de controle $C_{EXEC}^{\pi,\mathscr{A}}$ que será definida a seguir.

Inicialmente o ambiente $\mathscr{E}$ recebe alguma entrada, que representa o estado inicial do ambiente no qual a execução do protocolo se realiza. Em particular esta entrada representa todas as entradas externas do sistema, incluindo todas as entradas locais de todas as partes. A primeira ITI a ser chamada por $\mathscr{E}$ é definida pela função de controle para ser o adversário $\mathscr{A}$. Além disso, ao longo da computação, $\mathscr{E}$ pode chamar como subrotinas um

número ilimitado de ITIs, passar-lhes entradas, e obter saídas delas, sujeito à restrição que todas as ITIs chamadas tenham o mesmo SID, que por sua vez é escolhido por $\mathscr{E}$. o código destas ITIs é especificado pela função de controle para ser o código de π, independente do código especificado por $\mathscr{E}$. Consequentemente, todas as ITIs chamadas por $\mathscr{E}$, exceto $\mathscr{A}$, são partes de uma única instância de π. Por outro lado, $\mathscr{E}$ não pode passar entradas para, nem aceitar saídas de, nenhuma outra ITI. Em particular, $\mathscr{E}$ não pode interagir com subrotinas de partes de uma instância única de π.

A função de controle permite que partes e subpartes de π chamem outras ITIs e passem entradas e saídas para qualquer outra ITI que não seja o adversário ou o ambiente. Além disso, elas podem escrever mensagens na fita de comunicação de $\mathscr{A}$. Estas mensagens podem especificar uma identidade de uma parte ou subparte de π como destino final da mensagem.

Além disso, a função de controle permite que o adversário $\mathscr{A}$ envie mensagens para qualquer ITI no sistema. Nesse caso, diz-se que $\mathscr{A}$ entrega a mensagem. Não há necessariamente uma correspondência entre as mensagens enviadas pelas partes e as entregues pelo adversário, que não pode chamar subrotinas.

Execução do protocolo π com ambiente $\mathcal{E}$ e adversário $\mathcal{A}$

Dado um protocolo π, adversário $\mathcal{A}$ e ambiente $\mathcal{E}$ execute um sistema estendido de ITMs, conforme definido anteriormente, com ITM inicial $\mathcal{E}$ e uma função de controle como descrita abaixo. $\mathcal{E}$ inicia com parâmetro de segurança k e entrada x e prossegue como abaixo:

1. A primeira ITI a ser chamada por $\mathcal{E}$ é definida pela função de controle como sendo o adversário $\mathcal{A}$. O código de todas as outras ITIs chamadas por $\mathcal{E}$ é definido como sendo π. Além disso, todas essas ITIs devem ter o mesmo SID.

2. Uma vez que o adversário $\mathcal{A}$ é ativado, ele pode passar saída para $\mathcal{E}$ e também executar as seguintes ações:

   (a) $\mathcal{A}$ pode entregar uma mensagem m a uma parte ou subparte com identidade id; exceto para mensagens **`Corrupt`**, não há restrição quanto ao conteúdo ou identidade do destinatário.

   (b) $\mathcal{A}$ pode corromper uma ITI com identidade id, enviando uma mensagem **`Corrupt`** para a ITI em questão. Esta ação é permitida apenas se $\mathcal{A}$ recebeu de $\mathcal{E}$ uma mensagem ou entrada (**`Corrupt`**,id). A mensagem de $\mathcal{A}$ pode incluir parâmetros adicionais.

3. Uma vez que uma parte ou subparte de π é ativada, ela pode realizar as seguintes ações:

   - envia mensagens para $\mathcal{A}$,
   - passar entradas ou saídas para uma parte ou subparte de π
   - passar saídas para $\mathcal{E}$

   A ativação pode ser motivada por:

   - uma nova mensagem entregue por $\mathcal{A}$;
   - uma nova entrada vinda de $\mathcal{E}$;
   - ou um valor escrito na fita de saída de subrotina.

   A resposta a uma mensagem **`Corrupt`** pode variar de acordo com o modelo específico de corrupção e valores de parâmetros. No modelo de corrupção Bizantino, a parte reporta seu estado interno para o adversário. Em todas as ativações futuras por outras partes diferentes de $\mathcal{A}$, a parte corrompida envia seu estado local para $\mathcal{A}$ e segue as instruções de $\mathcal{A}$ na entrega de valores a outras partes.

O adversário $\mathcal{A}$ pode também corromper partes ou subpartes de π. Formalmente, a corrupção de uma parte ou subparte é modelada por uma mensagem especial **`Corrupt`** entregue por $\mathcal{A}$ para a ITI correspondente. A função de controle permite a entrega da mensagem somente se $\mathcal{A}$ recebeu anteriormente uma mensagem especial (**`Corrupt`**,id) do ambiente $\mathcal{E}$. Isso garante que o ambiente quais as partes que foram corrompidas.

A resposta de uma parte ou subparte a uma mensagem **Corrupt** não é definida no modelo geral, sendo sua definição deixada a cargo do protocolo. Isto permite capturar várias formas de corrupção no mesmo modelo computacional. Para prover uma possibilidade concreta de corrupção de uma parte ou subparte, é definido um possível modelo de corrupção, o Bizantino. Nele, uma vez que uma parte ou subparte recebe uma mensagem **Corrupt**, ela envia ao adversário $\mathscr{A}$ seu estado local corrente e em ativações futuras a parte corrompida segue as instruções de $\mathscr{A}$ independente de escritas externas por outras ITIs.

Deve-se dizer que o modelo permite que $\mathscr{A}$ chame novas ITIs entregando-lhes mensagens. Estas ITIs podem, mas não precisam, ser partes da instância de π com SID *sid*. Esta possibilidade é importante pois permite modelar instâncias de protocolos que são chamadas por mensagens vindas da rede. Poderia ser imposta a restrição permitindo $\mathscr{A}$ chamar apenas as partes da instância corrente. Ainda assim, seria necessário restringir que só as partes da instância de π com SID *sid* podem passar saídas para $\mathscr{E}$.

Sem perda de generalidade, os ambientes retorna apenas um bit de saída denotado pela variável aleatória $OUT_{\mathscr{E},C_{EXEC}^{\pi,\mathscr{A}}}(k,x)$, que também será referida como $EXEC_{\pi,\mathscr{A},\mathscr{E}}(k,x)$. O quadro acima resume o modelo de execução.

4.3. O framework UC

A seguir, o framework é apresentado, culminando com o Teorema da Composição na seção seguinte.

4.3.1. Breve descrição geral

Seguindo [Canetti 2000], o framework UC tem por base três componentes: o modelo real, que formaliza o processo de execução do protocolo na presença de um adversário, em um dado ambiente computacional; o modelo ideal, onde o processo que executa a funcionalidade ideal - aquela que o protocolo real pretende implementar - é formalizado. Neste processo as partes não se comunicam entre si, podendo apenas se comunicar com a funcionalidade ideal. Esta por sua vez captura os requisitos que o protocolo deve satisfazer, sendo tratada como uma terceira parte incorruptível; e a emulação da funcionalidade ideal pelo protocolo real. O protocolo real é dito ser seguro se suas funcionalidades correspondem àquelas prescritas pela funcionalidade ideal.

O protocolo real é representado por um conjunto de máquinas de Turing interativas - ITM, representando as partes envolvidas. Para que o protocolo seja realizável em condições realistas, as ITMs são limitadas em recursos, sendo assim probabilísticas e executando em tempo polinomial - PPT. Nas ITM, as fitas de entrada e saída modelam os valores de entrada e saída que são recebidos de, ou enviados para, outros programas rodando na mesma máquina, enquanto as fitas de comunicação modelam as mensagens recebidas e enviadas pelos outros participantes. O adversário também é modelado por uma ITM.

A rede de comunicação provida entre as partes é assíncrona, sem garantia de entrega das mensagens. Toda comunicação é pública, de forma que o adversário pode ter acesso a todo o tráfego, mas as mensagens trocadas são autenticadas, de forma que o ad-

versário não pode modificá-las. O adversário pode ser adaptativo na corrupção das partes e é ativo no controle destas. Tanto o adversário quanto o ambiente são PPT de forma que são realizáveis.

4.3.2. Execução do protocolo no modelo real

Seja um protocolo π, executado pelas partes $P_1, \ldots, P_n$, com um adversário $\mathscr{A}$ em um ambiente $\mathscr{E}$. Todas as partes tem um parâmetro de segurança $k \in \mathbf{N}$ e são polinomiais em k. A execução é uma sequência de ativações, onde a cada ativação um único participante ($\mathscr{E}$, $\mathscr{A}$ ou algum P_i) é ativado. O participante ativo lê a informação nas suas fitas de entrada (incluídas as de comunicação), executa código e escreve nas fitas de saída (também incluídas as de comunicação). O ambiente $\mathscr{E}$ pode além de isso ler e escrever nas fitas de entrada e de saída, respectivamente, das partes. O adversário $\mathscr{A}$ pode ler mensagens das fitas de saída das partes, copiá-las e entregá-las às partes destinatárias. Note que somente as mensagens geradas pelas partes podem ser entregues, já que o adversário $\mathscr{A}$ não pode alterar ou gerar duplicatas de mensagens. O adversário $\mathscr{A}$ pode corromper as partes, passando a controlá-la e a conhecer as informações internas por ela conhecidas.

Inicialmente, o ambiente $\mathscr{E}$ é ativado. Uma vez ativado, ele pode escreve informação na fita de entrada de alguma parte P_i ou do adversário $\mathscr{A}$. A ativação da entidade se dá assim que o ambiente $\mathscr{E}$ complete sua ativação, i.e. entre em um estado de espera. Se nenhuma fita de entrada for escrita a execução para. Quando uma parte P_i completa a ativação o ambiente $\mathscr{E}$ é ativado novamente. Sempre que o adversário $\mathscr{A}$ entrega uma mensagem a uma parte P_i, a parte é ativada em seguida, que ao se completar leva a nova ativação do ambiente $\mathscr{E}$. Se em alguma ativação o adversário $\mathscr{A}$ não entrega mensagem a alguma parte P_i, o ambiente $\mathscr{A}$ é ativado assim que o adversário $\mathscr{A}$ completar sua ativação. Note que este mecanismo permite que $\mathscr{E}$ e $\mathscr{A}$ troquem mensagens livremente usando suas fitas de entrada e saída, entre duas ativações de alguma parte P_i. A saída da execução do protocolo é a saída de $\mathscr{E}$, que sem perda de generalidade será um bit.

Por $REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x,\vec{r})$ entende-se a saída do ambiente $\mathscr{E}$ interagindo com o adversário $\mathscr{A}$ e partes P_i executando o protocolo π com parâmetro de segurança k, entrada x, e entrada aleatória $\vec{r} = r_{\mathscr{E}}, r_{\mathscr{A}}, r_1 \ldots, r_n$ (x e $r_{\mathscr{E}}$ para $\mathscr{E}$, $r_{\mathscr{A}}$ para $\mathscr{A}$; r_i para P_i). Por $REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x)$ entende-se a variável aleatória descrevendo $REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x,\vec{r})$ quando $\vec{r}$ é escolhido uniformemente. Por $REAL_{\pi,\mathscr{A},\mathscr{E}}$ entende-se a ensemble de distribuições de probabilidade $\{REAL_{\pi,\mathscr{A},\mathscr{E}}(k,x)\}_{k\in\mathbf{N},x\in\{0,1\}^*}$.

4.3.3. Execução no modelo ideal

A segurança de um protocolo é estabelecida comparando sua execução no modelo real com a de um processo ideal que realiza a tarefa desejada. Ao processo ideal está associada uma *funcionalidade ideal* que captura, i.e. especifica, a funcionalidade desejada. A funcionalidade ideal é modelada como mais uma ITM que interage com o ambiente e com o adversário como descrito a seguir.

Especificamente, o processo ideal envolve uma funcionalidade ideal $\mathscr{F}$, um processo adversário ideal $\mathscr{S}$, um ambiente $\mathscr{E}$ com entrada x e um conjunto de partes *dummy* $\tilde{P}_1, \ldots, \tilde{P}_n$. Cada parte *dummy* $\tilde{P}_i$ é modelada por uma ITM fixa e simples: sempre que uma $\tilde{P}_i$ recebe uma entrada, o valor é repassado para $\mathscr{F}$, copiando da fita de entrada para

a fita de comunicação; quando ativado por uma mensagem vida de $\mathscr{F}$, , a mensagem é copiada para a fita de saída. $\mathscr{F}$ recebe mensagens das partes *dummy* lendo das suas fitas de comunicação. $\mathscr{F}$ passa informações para as partes *dummy* enviando para suas fitas de comunicação. O processo adversário ideal $\mathscr{S}$ tem privilégios idênticos ao adversário no modelo real, exceto que não lhe é permitido acesso ao conteúdo das mensagens enviadas por $\mathscr{F}$ às partes *dummy*. $\mathscr{S}$ também pode corromper as partes *dummy*, controlar suas atividades futuras e tomar conhecimento da informação que elas possuam internamente.

A ordem dos eventos no processo ideal é idêntica à que ocorre no modelo real, com uma exceção. No modelo ideal, se uma parte *dummy* $\tilde{P}_i$ é ativada por um valor vindo do ambiente $\mathscr{E}$, este valor é copiado para a fita de comunicação da parte e o ambiente $\mathscr{E}$ é ativado em seguida. Uma vez que $\mathscr{F}$ tenha completado sua ação (possivelmente enviando uma mensagem para $\mathscr{S}$ ou algum $\tilde{P}_j$), a parte $\tilde{P}_i$ é ativada novamente. Deve-se ressaltar que no modelo ideal não há comunicação entre as partes *dummy*. A única comunicação que ocorre são as transferências entre as partes *dummy* e a funcionalidade ideal $\mathscr{F}$.

Por $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x,\vec{r})$ entende-se a saída do ambiente $\mathscr{E}$ interagindo com o adversário $\mathscr{S}$ e partes P_i executando o protocolo $\mathscr{F}$ com parâmetro de segurança k, entrada x, e entrada aleatória $\vec{r} = r_{\mathscr{E}}, r_{\mathscr{S}}, r_1 \ldots, r_n$ (x e $r_{\mathscr{E}}$ para $\mathscr{E}$, $r_{\mathscr{S}}$ para $\mathscr{S}$; r_i para P_i). Por $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x)$ entende-se a variável aleatória descrevendo $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x,\vec{r})$ quando $\vec{r}$ é escolhido uniformemente. Por $IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}$ entende-se a ensemble de distribuições de probabilidade $\left\{IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}(k,x)\right\}_{k\in\mathbf{N},x\in\{0,1\}^*}$.

4.3.4. Implementação segura de uma funcionalidade ideal

Um protocolo ρ é uma *implementação segura* (ou *realização segura*) de uma funcionalidade ideal $\mathscr{F}$ se para todo adversário $\mathscr{A}$ executando no modelo real existe um adversário $\mathscr{S}$ executando no modelo ideal, tal que nenhum ambiente $\mathscr{E}$, para qualquer valor de entrada, não lhe é possível distinguir com probabilidade não desprezível se está interagindo com $\mathscr{A}$ e as partes que executam ρ no modelo real, ou se está interagindo com $\mathscr{S}$ e a funcionalidade ideal $\mathscr{F}$ no modelo ideal.

Desta forma, para o ambiente, executar o protocolo ρ equivale a interagir com a funcionalidade ideal $\mathscr{F}$.

Definição 4.3.1. *Sejam $\xi = \{X(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ e $\psi = \{Y(k,a)\}_{k\in\mathbf{N},a\in\{0,1\}^*}$ duas ensembles de distribuições de probabilidade sobre $\{0,1\}$. Diz-se que ξ e ψ são indistinguíveis, denotado por $\xi \overset{c}{\approx} \psi$, se para todo $c \in \mathbf{N}$ existe um $k_0 \in \mathbf{N}$ tal que*

$$|PrX(k,a) = 1| - |PrY(k,a) = 1| < k^{-c}$$

para todo $k > k_0$ e todo a.

Definição 4.3.2. *Sejam $k, c \in \mathbf{N}$, $\mathscr{F}$ uma funcionalidade ideal e π um protocolo executado por n partes. Diz-se que π é uma implementação segura de $\mathscr{F}$ se para todo adversário $\mathscr{A}$ existe um adversário ideal $\mathscr{S}$ tal que para qualquer ambiente $\mathscr{E}$ temos que*

$$REAL_{\pi,\mathscr{A},\mathscr{E}} \overset{c}{\approx} IDEAL_{\mathscr{F},\mathscr{S},\mathscr{E}}.$$

4.4. O Teorema da Composição

Para apresentar o Teorema da Composição, é preciso inicialmente definir o modelo híbrido. No modelo híbrido é definido um modelo de computação em que um protocolo pode ter acesso, chamar, uma funcionalidade ideal.

4.4.1. O modelo híbrido

O modelo híbrido é idêntico ao modelo real, porém com algumas características a mais. Além de trocar mensagens entre si, as partes podem enviar e receber mensagens de um número ilimitado de cópias da funcionalidade ideal $\mathscr{F}$. Cada cópia de $\mathscr{F}$ é identificada por um identificador de sessão (SID) único. Todas as mensagens enviadas a uma dada cópia ou recebidas dela carregam o SID correspondente, que é escolhido pelo protocolo executado pelas partes.

A comunicação entre as partes e cada uma das cópias de $\mathscr{F}$ se dá como no modelo ideal. Assim, uma vez que alguma parte envia uma mensagem para alguma cópia de $\mathscr{F}$, esta cópia é imediatamente ativada e lê a mensagem da fita da parte. Além disso, neste modelo, apesar do adversário ser responsável por entregar as mensagens vindas das cópias de $\mathscr{F}$ para as partes, ele não tem acesso ao conteúdo das mensagens. Deve-se ressaltar que o ambiente não tem acesso direto às cópias de $\mathscr{F}$. Isso decorre da definição de segurança envolvendo o modelo híbrido, que será baseada na incapacidade do ambiente de distinguir entre o modelo real e o híbrido.

Já se pode definir como uma chamada à funcionalidade ideal $\mathscr{F}$ é substituída por uma chamada ao protocolo. Seja π um protocolo no modelo híbrido, e seja ρ um protocolo é uma implementação segura de uma funcionalidade ideal $\mathscr{F}$, com respeito a alguma classe de adversários. O protocolo composto π^ρ é construído substituindo o código de cada ITM de π deforma que a primeira mensagem enviada a cada cópia de $\mathscr{F}$ é substituída por uma chamada a uma nova cópia de ρ, com novos valores aleatórios de entrada, e com o conteúdo da mensagem como entrada. Cada mensagem subsequente para a cópia de $\mathscr{F}$ é substituída pela cópia correspondente de ρ, com o conteúdo passado para ρ como nova entrada. Cada valor de saída gerado pela cópia de ρ é tratado como uma mensagem recebida da cópia correspondente de $\mathscr{F}$.

4.4.2. Enunciado do Teorema

Na sua forma geral, o Teorema da Composição diz que se ρ é uma implementação segura de $\mathscr{F}$ então a execução do protocolo composto π^ρ no modelo real "emula" uma execução de $\pi^\mathscr{F}$, em que π chama cópias de $\mathscr{F}$, no modelo híbrido. Assim, para todo adversário $\mathscr{A}$ no modelo real existe um adversário $\mathscr{H}$ no modelo híbrido tal que nenhum ambiente $\mathscr{E}$ consegue distingue com probabilidade não desprezível se está interagindo com $\mathscr{A}$ e π^ρ no modelo real ou com $\mathscr{H}$ e $\pi^\mathscr{F}$ no modelo híbrido.

Teorema 4.4.1. (Teorema da Composição) *Seja $\mathscr{F}$ uma funcionalidade ideal, e π um protocolo, que composto com $\mathscr{F}$ resulta em $\pi^\mathscr{F}$, no modelo híbrido. Seja ρ um protocolo que é uma implementação segura de $\mathscr{F}$. Então para todo adversário $\mathscr{A}$ no modelo real existe um adversário $\mathscr{H}$ no modelo híbrido tal que para qualquer ambiente $\mathscr{E}$ temos que*

$$REAL_{\pi^\rho,\mathscr{A},\mathscr{E}} \overset{c}{\approx} HIBRIDO_{\pi^\mathscr{F},\mathscr{H},\mathscr{E}}$$

Uma formulação mais específica do resultado geral afirma que se π é uma implementação segura de uma funcionalidade ideal $\mathcal{G}$. $\mathcal{G}$ composto com a funcionalidade $\mathcal{F}$ resulta em $\mathcal{G}^{\mathcal{F}}$ equivale a $\pi^{\mathcal{F}}$ no modelo híbrido. Seja ρ um protocolo que é uma implementação segura de $\mathcal{F}$. Então π^{ρ} é uma implementação segura de $\mathcal{G}^{\mathcal{F}}$ no modelo real.

Teorema 4.4.2. *Sejam $\mathcal{F}$ e $\mathcal{G}$ funcionalidades ideais. Seja π um protocolo de n partes que é uma implementação segura de $\mathcal{G}$ no modelo híbrido e seja ρ um protocolo de n partes que é uma implementação segura de $\mathcal{F}$. Então o protocolo π^{ρ} é uma implementação segura de $\mathcal{G}^{\mathcal{F}}$.*

A prova detalhada se encontra em [Canetti 2001]. Entretanto, as principais ideias serão apresentadas na sequência. Sejam π e ρ protocolos e $\mathcal{F}$ uma funcionalidade ideal, sendo ρ uma implementação segura de $\mathcal{F}$. Seja $\mathcal{A}$ o adversário para π^{ρ} no modelo real. Deseja-se construir um adversário $\mathcal{A}'$ para $\pi^{\mathcal{F}}$ no modelo híbrido tal que $\mathcal{E}$ não conseguirá diferenciar se está interagindo com π^{ρ} e $\mathcal{A}$ no modelo real ou com $\pi^{\mathcal{F}}$ e $\mathcal{A}'$ no modelo híbrido. Para isto, será usado um adversário $\mathcal{S}$, um simulador, que atual sobre uma única instância de $\mathcal{F}$. Essencialmente, $\mathcal{A}'$ executará uma versão simulada de $\mathcal{A}$, seguindo suas instruções. As interações de $\mathcal{A}$ com as várias instâncias de ρ serão simuladas usando várias instâncias de $\mathcal{S}$. Assim, $\mathcal{A}'$ fará o papel do ambiente para as várias instâncias de $\mathcal{S}$. A habilidade de $\mathcal{A}'$ conseguir informações as múltiplas instâncias de $\mathcal{S}$ fazendo o papel do ambiente, é o principal ponto da prova.

A validade da simulação como um todo é demonstrada reduzindo-a à validade de $\mathcal{S}$. O tratamento das diversas instâncias de $\mathcal{S}$ é feito usando um argumento híbrido, que define várias execuções híbridas, nas quais um certo número de instâncias de $\mathcal{F}$ são substituídas por instâncias de ρ. este argumento híbrido é possível pelo fato de $\mathcal{S}$ ser definido independentemente do ambiente. Assim, não muda sob os vários ambientes híbridos.

Na prova em si, a questão referente à construção de um adversário no modelo ideal para todo adversário no modelo real é simplificada. ao invés de se quantificar sobre todos os adversários possíveis, recorre-se à utilização de um adversário *dummy*. Desta forma, é suficiente exigir que o adversário ideal $\mathcal{S}$ seja capaz de simular este adversário *dummy*, que é bem simples. Este adversário *dummy* pode apenas receber todas as mensagens geradas pelo ambiente e repassá-las às partes que são destinatárias. Assim, o adversário *dummy*, dado seu pequeno poder, é o mais difícil de simular pois simulá-lo indica que se pode simular qualquer outro. Intuitivamente, o adversário *dummy* é o mais difícil de simular porque dá ao ambiente controle total sobre as comunicações, deixando pouco espaço de manobra para o simulador.

Especificamente, segue a descrição do adversário *dummy* $\mathcal{D}$. Quando ativado com uma mensagem *m* na sua fita de entrada, $\mathcal{D}$ passa *m* para o ambiente $\mathcal{E}$ (note que a mensagem já inclui a identidade do destinatário). Quando ativado com uma entrada (m, id, c) do ambiente $\mathcal{E}$, onde *m* é uma mensagem, *id* é um destinatário, e *c* um código, $\mathcal{D}$ entrega a mensagem *m* à parte *id* (o código *c* é usado no caso de não existir uma parte com identidade *id*). Assim, $\mathcal{D}$ é capaz de corromper partes quando instruído por $\mathcal{E}$, e coletar informação para *E*.

4.5. Comprometimento de Bit - Bit Commitment BC

Comprometimento (*commitment*) é uma das primitivas criptográficas mais importantes, sendo utilizado na construção de vários outros protocolos criptográficos como zero knowledge (por exemplo [Goldreich et al. 1987], [Brassard et al. 1988], [Damgård 1990]), protocolos gerais de avaliação de funções (por exemplo [Goldreich et al. 1987], [Galil et al. 1988]) e vários outros. Os protocolos de comprometimento em si tem sido objeto de vários estudos como por exemplo [Blum 1982], [Naor 1991], [Dolev et al. 2000], [Naor et al. 1998], [Beaver 1996], [Di Crescenzo et al. 1998], [Fischlin and Fischlin 2000].

A ideia básica da noção de comprometimento é bem simples: uma parte, a que se compromete, entrega ao receptor o equivalente digital de um envelope lacrado contendo um valor x. desse ponto em diante, a parte comprometida não pode alterar o valor dentro do envelope, e enquanto esta parte não passar ao receptor instruções para abrir o envelope, este não tem conhecimento algum sobre o valor de x. O primeiro requisito, que impede que a parte que se compromete altere o valor de x antes da abertura é referida como *binding* enquanto o requisito do receptor só poder tomar conhecimento do valor de x após a abertura pela parte comprometida é referida como *hiding*

Entretanto, a formalização dessa ideia intuitiva não é trivial. O tratamento normalmente dado [Goldreich 2000] se revela insuficiente para algumas aplicações, basicamente por tratar o comprometimento de forma isolada. Assim, em geral, não se garante a segurança quando um protocolo de comprometimento é usado como componente em outros protocolos, ou quando múltiplas de suas cópias executam concorrentemente.

A seguir, o conceito de comprometimento será utilizado para demonstrar duas aplicações do *framework* UC. Inicialmente, o conceito será descrito de forma abstrata como uma funcionalidade ideal. Será demonstrado que como definida, não é possível implementar seguramente (UC segura) a funcionalidade sem que se recorra a hipóteses adicionais. Na sequência, será construído um protocolo de comprometimento UC seguro que inclui uma hipótese de setup. Tanto a impossibilidade da implementação da funcionalidade ideal "pura´´ quanto à segurança UC do protocolo "real´´ são demonstradas.

4.5.1. A Funcionalidade Ideal $\mathscr{F}_{\mathbf{BC}}$

A funcionalidade ideal que captura os requisitos esperados sobre o comprometimento de bit é apresentada a seguir. Na funcionalidade é capturada a ideia de envelope presente na motivação inicial do BC. Como enunciada, a funcionalidade trata do caso de um comprometimento único, podendo ser facilmente estendida para lidar com o caso geral de múltiplos comprometimentos. Outro ponto a destacar é que a funcionalidade lida com o comprometimento de bit, podendo também ser estendida para o caso de *strings* aplicando o Teorema da Composição, sendo também possível estender a funcionalidade diretamente, sem uso do Teorema da Composição. Os protocolos de comprometimento de *string* resultantes seriam certamente realizações mais eficientes da funcionalidade proposta.

A Funcionalidade Ideal $\mathscr{F}_{\mathbf{BC}}$ procede como descrito a seguir. A fase de comprometimento é modelada com a funcionalidade inicialmente recebendo uma mensagem

$(\texttt{Commit}, sid, P_i, P_j, b)$ de P_i, a parte que se compromete. Onde o *sid* é um identificador de sessão usado para diferenciar entre as instâncias da funcionalidade que estejam executando simultaneamente, P_j é a parte receptora, e $b \in \{0,1\}$ é o valor com o qual P_i se comprometeu. Na resposta, a funcionalidade precisa dar conhecimento à outra parte P_n e ao adversário ideal $\mathscr{S}$ de que P_i se comprometeu a algum valor de b, associado À sessão *sid*. Isto é feito enviando a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$, para P_j e $\mathscr{S}$. A fase de abertura é iniciada com P_i enviando a mensagem $(\texttt{Open}, sid, P_i, P_j)$ para $\mathscr{F}_{\mathbf{BC}}$, que por sua vez envia a mensagem $(\texttt{Open}, sid, P_i, P_j, b)$ para P_j e $\mathscr{S}$, completando a execução.

A figura abaixo resume a descrição da funcionalidade:

Funcionalidade Ideal $\mathscr{F}_{\mathbf{BC}}$

$\mathscr{F}_{\mathbf{BC}}$ é definida como segue, executando com as partes $P_1, \ldots, P_n$, na presença de um adversário $\mathscr{S}$:

1. Ao receber uma mensagem $(\texttt{Commit}, sid, P_i, P_j, b)$ de P_i, $b \in \{0,1\}$:
   - armazene o bit b e
   - envie a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$, para P_j e $\mathscr{S}$.

   Ignore mensagens **Commit** subsequentes.
2. Ao receber uma mensagem $(\texttt{Open}, sid, P_i, P_j)$ de P_i, proceda da seguinte forma:
   - se o bit b estiver armazenado,
     - então envie a mensagem $(\texttt{Open}, sid, P_i, P_j, b)$ para P_j;
     - caso contrário, pare a execução.

Vale ressaltar que como definida, a funcionalidade $\mathscr{F}_{\mathbf{BC}}$ não permite a cópia de comprometimento. Suponha que A se compromete com valor x para alguma parte B, e que o protocolo permitisse que B se comprometesse com o mesmo valor para uma parte C, simplesmente repassando o conteúdo da mensagem enviada por A. Tal protocolo não seria uma realização segura da funcionalidade $\mathscr{F}_{\mathbf{BC}}$. este requisito pode parecer difícil de ser satisfeito pois B pode sempre atuar como '*man-in-the-middle*. Deve-se lembrar que as identidades são únicas uma vez que se assume que as comunicações são autenticadas.

4.5.2. Impossibilidade de segurança UC para BC sem hipótese de setup

Esta seção ilustra uma das aplicações do framework UC. Em particular será apresentada sua aplicação para análise de um protocolo, no caso o comprometimento de bit BC. Será mostrado nesta sessão que tal como definida, sem acessar alguma outra funcionalidade ideal, $\mathscr{F}_{\mathbf{BC}}$ não é UC segura. Ou seja, não existe protocolo que seja uma implementação segura sem envolver uma terceira parte na interação e permitir a execução com sucesso quando tanto o emissor e o receptor são honestos. A impossibilidade se mantem mesmo

sob um requisito mais fraco em que para qualquer adversário real e qualquer ambiente deve existir um adversário ideal, onde o simulador pode depender do ambiente.

Esta situação em que uma funcionalidade não pode chamar outra, será referida como modelo simples. Note que existem protocolos de BC que são UC no modelo simples, se fizer uso de terceiras partes, ou servidores, se a maioria destes se mantiver não corrompida. Isto segui do resultado geral de [C01] que mostra que praticamente qualquer funcionalidade pode ser realizada neste arranjo.

Diz-se que o protocolo π com n partes $P_1, \ldots, P_n$ é *bilateral* se apenas duas dentre as n partes trocam mensagens entre si. Um protocolo bilateral de comprometimento π *termina com respeito à execução* se com probabilidade não desprezível, o receptor honesto P_j aceita um comprometimento de um emissor honesto P_i e envia a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$; além disso, quando o receptor honesto recebe uma mensagem válida de abertura do emissor honesto, com valor m e identificador de sessão sid, ele envia a mensagem $(\texttt{Open}, sid, P_i, P_j, m)$ com probabilidade não desprezível.

Teorema 4.5.1. *Não existe protocolo bilateral de comprometimento π convergente com respeito à execução que seja uma implementação segura da funcionalidade $\mathcal{F}_{\mathbf{BC}}$ no modelo simples, mesmo que se permita que o adversário ideal $\mathcal{S}$ dependa do ambiente $\mathcal{E}$.*

Prova: A ideia da prova é a que segue. Considere uma execução no modelo real do protocolo entre uma parte P_i que se compromete com um valor b e o receptor P_j. Suponha que P_i foi corrompido pelo adversário $\mathcal{A}$ e é por ele controlado, enquanto o receptor P_j se mantem honesto. Assuma também que o adversário $\mathcal{A}$ apenas envia as mensagens geradas pelo ambiente $\mathcal{E}$ e repassa a este as mensagens enviadas a P_i. No início do processo, o ambiente escolhe aleatoriamente um bit b, mantendo seu valor em segredo, e gera as mensagens para P_i executando o protocolo para uma parte honesta, que se compromete com b, e as respostas de P_j.

Para simular o comportamento acima descrito, no modelo ideal, o adversário $\mathcal{S}$ deve ser capaz de prover à funcionalidade ideal um valor para o bit comprometido. Ou seja, o ideal $\mathcal{S}$ deve extrair o bit das mensagens geradas pelo ambiente, sem que "rebobine", i.e. volte o estado de execução do ambiente. Entretanto, será mostrado, se o esquema de comprometimento de bit, permite que o simulador extraia com sucesso o bit comprometido, então o esquema não é seguro, pois um receptor corrompido conseguiria obter o valor do bit comprometido interagindo com um emissor honesto.

Mais precisamente, no modelo real, seja o protocolo bilateral π executado com emissor P_i e receptor P_j. Considere o ambiente $\mathcal{E}$ e o adversário real $\mathcal{A}$. No início da execução o adversário $\mathcal{A}$ corrompe P_i, a parte que se compromete no protocolo. Assim, $\mathcal{A}$ faz com que P_i envie toda mensagem recebida de $\mathcal{E}$ e reporta toda resposta recebida por P_j de $\mathcal{E}$.

O ambiente $\mathcal{E}$ escolhe aleatoriamente um bit b, mantendo seu valor em segredo, e gera as mensagens para P_i executando o protocolo para uma parte honesta, que se compromete com b e as respostas de P_j e segue o programa do emissor honesto que se compromete com b, conforme especificado por π. Uma vez que o receptor honesto envia a mensagem `Receipt`, o ambiente faz com que $\mathcal{A}$ abra o comprometimento com valor b.

Uma vez que o receptor envia a mensagem $(\texttt{Open}, sid, P_i, P_j, b')$, o ambiente gera o valor 1 se $b = b'$ e 0 caso contrário.

Já no modelo ideal, uma vez que o receptor envia uma mensagem **Receipt** antes de se iniciar a fase de abertura, o adversário ideal $\mathcal{S}$ para o para $\mathcal{A}$, $\mathcal{E}$ deve enviar a mensagem $(\texttt{Commit}, sid, P_i, P_j, b)$ para funcionalidade ideal $\mathcal{F}_{\mathbf{BC}}$ *antes* de conhecer o valor do bit comprometido b no final da fase de abertura. Entretanto, o receptor honesto envia o valor b' recebido de $\mathcal{F}_{\mathbf{BC}}$ na fase de abertura. Isso implica que, para ter sucesso, $\mathcal{S}$ deve obter o valor correto do bit comprometido b já na fase inicial de comprometimento, o que viola o sigilo do protocolo de comprometimento.

Formalizando, suponha que existe um adversário ideal $\mathcal{S}$ tal que:

$$REAL_{\pi,\mathcal{A},\mathcal{E}} \overset{c}{\approx} IDEAL_{\mathcal{F},\mathcal{S},\mathcal{E}}$$

Será construído um novo ambiente $\mathcal{E}'$ e um novo adversário real $\mathcal{A}'$ para o qual não existe um adversário ideal apropriado para π. nessa nova situação, $\mathcal{A}'$ corrompe o receptor P_j no começo da execução do protocolo. Durante a execução, $\mathcal{A}'$ obtém as mensagens do emissor honesto P_i, passando-as a uma cópia virtual de $\mathcal{S}$. As respostas de $\mathcal{S}$, como se geradas pelo receptor honesto, são encaminhadas a P_i em nome da parte corrompida P_j. Em dado momento, $\mathcal{S}$ envia a mensagem de comprometimento $(\texttt{Commit}, sid, P_i, P_j, b')$ para $\mathcal{F}_{\mathbf{BC}}$. Então o adversário $\mathcal{A}'$ gera o valor de saída b' e para.

O ambiente $\mathcal{E}'$ ordena que a parte honesta se comprometa a um bit b escolhido aleatoriamente e cujo valor é mantido em segredo. Note que em nenhum momento se dá a abertura. O ambiente $\mathcal{E}'$ então conclui gerando o valor 1 se e sós se o valor de saída b' gerado pelo adversário $\mathcal{A}'$ satisfaz $b = b'$.

Pela propriedade de terminação definida acima, obtém-se do simulador $\mathcal{S}$ um bit b' com probabilidade não desprezível. Este bit b' é uma boa aproximação do bit b, já que $\mathcal{S}$ simula o protocolo π com erro desprezível. Consequentemente, a probabilidade do bit b gerado por $\mathcal{A}'$ estar correto é 1/2 mais um valor não desprezível. Porém, é impossível que um adversário ideal $\mathcal{S}'$ consiga acertar o valor de b probabilidade não desprezível maior que 1/2, uma vez que a visão de $\mathcal{S}'$ no modelo ideal é estatisticamente independente do bit b, lembrando que o comprometimento com b não é aberto. □

4.6. Um protocolo UC seguro para comprometimento de bit

Como estabelecido na seção anterior, vimos a impossibilidade de implementação segura (UC segura) da funcionalidade ideal $\mathcal{F}_{\mathbf{BC}}$ por um protocolo que não recorra a pelo menos mais outra funcionalidade. A seguir será apresentada um funcionalidade que auxiliará na construção de um protocolo para BC que é UC seguro, seguido de esquema de comprometimento, que será demonstrado ser UC seguro.

4.6.1. A Funcionalidade Ideal $\mathcal{F}_{\mathbf{CRS}}$

A funcionalidade que será utilizada juntamente com $\mathcal{F}_{\mathbf{BC}}$ para a construção do protocolo UC seguro é descrita a seguir. A funcionalidade ideal $\mathcal{F}_{\mathbf{CRS}}$, onde CRS é uma *string* comum de referência, consiste na distribuição de uma *string* entre todas as partes envolvidas. A *string* é distribuída antes do início da interação entre as partes, sendo escolhida

segundo uma distribuição de probabilidade especificada.

Funcionalidade Ideal $\mathcal{F}_{\mathbf{CRS}}$

$\mathcal{F}_{\mathbf{CRS}}$ é definida como segue, parametrizada por uma distribuição D:

1. Quando ativada pela primeira vez, com entrada (valor, *sid*):
   - escolha um valor $d \xleftarrow{R} D$
   - e envie d para a parte que a ativou.
2. Nas ativações seguintes, envie d para a parte que a ativou.

Note que na definição acima, a funcionalidade está parametrizada por uma distribuição de probabilidade D. Como definida a funcionalidade $\mathcal{F}_{\mathbf{CRS}}$ já está adequada ao modelo híbrido, onde se pode ter acesso à funcionalidade ideal.

Como definida, $\mathcal{F}_{\mathbf{CRS}}$ tem algumas características:

- no modelo real, as partes tem acesso a uma *string* comum e pública que é escolhida previamente de acordo com uma distribuição especificada pelo protocolo executado pelas partes.
- no modelo ideal, pode ser que uma funcionalidade não faça uso de $\mathcal{F}_{\mathbf{CRS}}$, como é o caso de $\mathcal{F}_{\mathbf{BC}}$. assim, um adversário ideal que opere simulando um adversário real pode fazer as vezes de $\mathcal{F}_{\mathbf{CRS}}$ para o adversário simulado. Isto significa que o adversário ideal pode escolher a *string* comum arbitrariamente.

Vale ressaltar que por não usar o *string* comum no modelo ideal, a validade do que acontece no modelo ideal não é afetada já que $\mathcal{F}_{\mathbf{CRS}}$ estaria executando no modelo híbrido. Assim, a validade da noção de segurança se mantem.

Do ponto de vista da composição, é preciso deixar claro alguns aspectos acerca de $\mathcal{F}_{\mathbf{CRS}}$. Cada cópia de ρ no protocolo composto π^{ρ} deve ter sua própria cópia da *string* de referência, i.e. chama uma instância própria de $\mathcal{F}_{\mathbf{CRS}}$. Se isto não for o caso, o Teorema da Composição não poderá ser aplicado.

4.6.2. O Esquema de Comprometimento de Bit com hipótese de setup

O protocolo apresentado a seguir é uma implementação segura da funcionalidade $\mathcal{F}_{\mathbf{BC}}$ se baseia em permutação trapdoor e utiliza cada *string* comum uma única vez, em um único comprometimento. A construção se baseia no esquema proposto em [Di Crescenzo et al. 1998], que por sua vez é uma modificação do esquema apresentado em [Naor 1991].

A ideia do protocolo é a que segue. Seja G um gerador pseudo aleatório que expande n bits de entrada em $4n$ bits de saída. Para o parâmetro de segurança n, o receptor

envia um *string* aleatório σ ao emissor, que escolhe um $r \in \{0,1\}^n$ que computa $G(r)$ e retorna $G(r)$ ou $G(r) \oplus \sigma$ para se comprometer com 0 ou 1 respectivamente. Na abertura, o emissor envia b e r. Pela pseudo aleatoriedade de G o receptor não pode distinguir os dois casos e com probabilidade 2^{-2n} sobre a escolha de r é impossível encontrar r_0 e r_1 tais que:

$$G(r_0) = G(r_1) \oplus \sigma$$

Em [Di Crescenzo et al. 1998] uma versão equivocável do esquema em [Naor 1991] foi proposta. Suponha que σ não é escolhida pelo receptor mas que seja uma comum aleatória. Então, se $\sigma = G(r_0) \oplus G(r_1)$ para r_0 e r_1 aleatórios, e o emissor entrega $G(r_0)$ ao receptor, será mais fácil abrir o comprometimento como 0 para r_0 e 1 para r_1, já que $G(r_0) \oplus \sigma = G(r_1)$. Por outro lado, a escolha de σ daquela forma é indistinguível de uma verdadeira escolha aleatória.

O protocolo UC seguro π_{BC}^{CRS}, descrito a seguir, parte da ideia do esquema acima usando um gerador pseudo aleatório com uma propriedade *trapdoor*. Será usado um gerador Blum-Micali-Yao, porém com permutações *trapdoor* ao invés de permutações *one-way* [Yao 1982] [Blum and Micali 1984]. Seja *KGen* um algoritmo eficiente que com entrada 1^n gera uma chave pública *pk* e um *trapdoor td*. A chave *pk* descreve uma permutação *trapdoor* f_{pk} sobre $\{0,1\}^n$.

Antes de definir o gerador, algumas definições iniciais são necessárias. O primeiro conceito é o de função one-way, que captura a noção intuitiva de uma função não inversível.

Definição 4.6.1. *Uma função $f : \{0,1\}^* \to \{0,1\}^*$, é one-way se:*

- *existe um algoritmo que para a entrada x computa $f(x)$ em tempo polinomial*
- *para todo algoritmo PPT $\mathscr{A}$, a probabilidade de $\mathscr{A}$ ser a inversa de $f(x)$*

$$Pr[f(A(f(x))) = f(x)]$$

é desprezível.

Outro conceito necessário é o de permutação trapdoor, que captura a noção de uma função one-way que com alguma informação adicional se torna fácil inverter.

Definição 4.6.2. *Uma permutação trapdoor é definida pela tripla (G,E,D) tal que:*

1. *G é um algoritmo que para a entrada 1^k gera um par de chaves (pk,td)*
2. *E é um algoritmo determinístico tal que para todo pk, o mapeamento $x \to E(pk,x)$ é uma bijeção*
3. *D é um algoritmo determinístico tal que para todos os pares de chaves possíveis (pk,td) gerados por G, e para todo x:*

$$D(td,E(pk,x)) = x \tag{1}$$

Definição 4.6.3. *Um predicado hardcore B(.) de uma função one-way $f(x)$, é uma função cuja saída é um único bit para qual não existe algoritmo PPT que compute $B(f(x))$ com probabilidade significativamente maior que 1/2 sobre uma escolha aleatória de x.*

Assim, já se pode definir o gerador[1].

Definição 4.6.4. *Seja $B(.)$ um predicado hardcore para f_{pk}. Define-se o gerador que expande n bits de entrada em $4n$ bits de saída com a descrição pública pk como:*

$$G_{pk}(r) = (B(f_{pk}^{(4n)}(r)), B(f_{pk}^{(4n-1)}(r)), \ldots, B(f_{pk}(r)), B(r))$$

onde $f_{pk}^{(i)}(r)$ é a i-ésima aplicação de f_{pk} a r.

Uma importante propriedade deste gerador é que dado um *trapdoor td* para *pk* é fácil dizer se um dado $y \in \{0,1\}^{4n}$ está na imagem de G_{pk}. O *string* público aleatório no esquema π_{BC}^{CRS} consiste em um *string* aleatório de $4n$ bits, junto com duas chaves públicas pk_0 e pk_1 que descrevem os geradores pseudo aleatórios *trapdoor* G_{pk_0} e G_{pk_1}, que expandem n bits de entrada em $4n$ bits de saída. As chaves públicas pk_0 e pk_1 são geradas por duas execuções independentes do algoritmo de geração de chaves *KGen* com entrada 1^n, juntamente com os *trapdoors* correspondentes td_0 e td_1 respectivamente.

Para se comprometer com um bit $b \in \{0,1\}$, o emissor escolhe um *string* aleatório $b \in \{0,1\}^n$, calcula $G_{pk_b}(r)$ e $y = G_{pk_b}(r)$ se $b = 0$, ou $y = G_{pk_b}(r) \oplus \sigma$ se $b = 1$. Na abertura, o emissor envia (b,r) ao receptor, que verifica se $y = G_{pk_b}(r)$ para $b = 0$, ou se $y = G_{pk_b}(r) \oplus \sigma$ para $b = 1$.

Claramente, o esquema abaixo satisfaz as condições de *hiding* e *binding*: a primeira computacionalmente e a segunda estatisticamente. Em uma simulação, σ assume o valor $\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$. Assim, transmitindo $y = G_{pk_0}(r_0)$ posteriormente pode-se abrir esse valor com 0 enviando r_0 e com 1 enviando r_1.

Além disso, caso se conheça td_0 e pk_0, um *string* y^* gerada pelo adversário pode ser checada se está na imagem de G_{pk_0} representando um comprometimento com o valor 0. A menos que y^* esteja na imagem de G_{pk_0} e simultaneamente $y^* \oplus \sigma$ pertença à imagem de G_{pk_1}, o valor extraído do bit comprometido é único e estará correto com respeito a td_0.

[1] para maiores detalhes e uma definição formal de geradores pseudo aleatórios vide apêndice

Esquema de Comprometimento de Bit com hipótese de setup - π_{BC}^{CRS}

- *string pública:*
  - σ - *string* aleatória em $\{0,1\}^{4n}$
  - pk_0, pk_1 - chaves para os geradores $G_{pk_0}, G_{pk_1} : \{0,1\}^n \to \{0,1\}^{4n}$
- *comprometimento para o bit $b \in \{0,1\}$ com SID sid:*
  - emissor P_i
    - calcula $G_{pk_b}(r)$
    - atribui $y = G_{pk_b}(r)$ se $b = 0$, ou $y = G_{pk_b}(r) \oplus \sigma$ se $b = 1$
    - envia mensagem $(\texttt{Commit}, sid, P_i, P_j, y)$ para o receptor P_j
  - receptor P_j
    - ao receber a mensagem $(\texttt{Commit}, sid, P_i, P_j, y)$ do emissor P_i, envia de volta a mensagem $(\texttt{Receipt}, sid, P_i, P_j, b)$
- *abertura para y*
  - emissor P_i
    - envia (b, r) ao receptor
  - emissor P_i
    - verifica se $y = G_{pk_b}(r)$ para $b = 0$, ou se $y = G_{pk_b}(r) \oplus \sigma$ para $b = 1$
    - se a verificação tem sucesso, gera a mensagem $(\texttt{Open}, sid, P_i, P_j, b)$

4.6.3. Segurança UC do protocolo BC com CRS - π_{BC}^{CRS}

Como visto, o esquema π_{BC}^{CRS} suporta a equivocabilidade e a extratibilidade. Já se pode então estabelecer que o esquema de protocolo π_{BC}^{CRS} é UC seguro, i.e. π_{BC}^{CRS} é uma implementação segura da funcionalidade ideal $\mathscr{F}^{\mathbf{BC}}$.

Teorema 4.6.1. *O protocolo π_{BC}^{CRS} é uma implementação segura da funcionalidade ideal $\mathscr{F}^{\boldsymbol{BC}}$ acessando a funcionalidade ideal $\mathscr{F}^{\boldsymbol{CRS}}$.*

Prova: A prova está dividida em partes. Inicialmente, será descrito o adversário ideal, o simulador $\mathscr{S}$. Definido $\mathscr{S}$, será tratada a questão da indistinguibilidade, que compreende três situações distintas. As questões relativas à indistinguibilidade serão reduzidas à distinção entre uma sequência realmente aleatória e outra pseudo aleatória.

Simulação

O adversário ideal $\mathscr{S}$ é descrito a seguir. $\mathscr{S}$ executa juntamente com $\mathscr{E}$ e, em para-

lelo executa uma cópia virtual do adversário real $\mathscr{A}$, que é executado como uma caixa preta. Isto é, $\mathscr{S}$ age como uma interface entre $\mathscr{A}$ e $\mathscr{E}$ emulando uma cópia de uma execução real de π_{BC}^{CRS} para $\mathscr{A}$, incorporando as interações de $\mathscr{E}$ no modelo ideal e, vice-versa, encaminhando as mensagens de $\mathscr{A}$ para $\mathscr{E}$.

1. No início, o simulador $\mathscr{S}$ prepara σ selecionando os pares de chaves

$$\begin{aligned}(pk_0, td_0) &\leftarrow KGen(1^n),\\ (pk_1, td_1) &\leftarrow KGen(1^n)\end{aligned}$$

   atribuindo a σ o valor

$$\sigma = G_{pk_0}(r_0) \oplus G_{pk_1}(r_1)$$

   para $r_0, r_1 \in \{0,1\}^n$ aleatórios.

   Esta *string* σ será chamada de *falsa* com respeito aos valores $pk_0, pk_1, G_{pk_0}(r_0)$ e $G_{pk_1}(r_1)$.

   Na sequência, inicia a execução da simulação de $\mathscr{A}$ e da execução com $\mathscr{E}$ usando a *falsa* σ e pk_0, pk_1.

2. Se em algum ponto da execução:

   - o ambiente $\mathscr{E}$ escreve a mensagem $(\texttt{Commit}, sid, P_i, P_j, b)$ na fita da parte não corrompida $\tilde{P}_i$,
   - e este a copia para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$.

   Então o simulador $\mathscr{S}$, que não pode ler o valor do bit mas toma conhecimento de que houve um comprometimento recebendo a mensagem $(\texttt{Receipt}, sid, P_i, P_j)$,

   - informa $\mathscr{A}$ que P_i enviou a P_j

$$y = G_{pk_0}(r_0)$$

   .

3. Se em algum ponto da execução:

   - o ambiente $\mathscr{E}$ ordena que a parte não corrompida $\tilde{P}_i$ realize a abertura
   - e esta parte, corretamente, se comprometeu com algum bit b, cujo valor foi mantido secreto

   Então o adversário ideal $\mathscr{S}$ deve enviar o valor

$$y = G_{pk_0}(r_0)$$

   fazendo as vezes da parte P_i na simulação (em modo caixa preta) do adversário $\mathscr{A}$.

4. Se o adversário $\mathscr{A}$ simulado

- permite que alguma parte corrompida P_i envie a mensagem $(\texttt{Commit}, sid, y^*)$ a alguma parte honesta P_j
- então $\mathscr{S}$ verifica com o auxílio do *trapdoor* td_0 se y^* está na imagem de G_{pk_0} ou não.

Se for o caso,

- envia a mensagem $(\texttt{Commit}, sid, P_i, P_j, 0)$ para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$
  - como se fosse a parte P_i, para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$; caso contrário, $\mathscr{S}$ envia a mensagem $(\texttt{Commit}, sid, P_i, P_j, 1)$

5. Se $\mathscr{A}$ ordena:

   - que alguma parte corrompida P_i realize a abertura com y^* válido e bit b^* correto,
   - então $\mathscr{S}$ compara com o valor de b^* ao valor previamente extraído e para a execução se eles diferem.
   - caso contrário,
     - $\mathscr{S}$ envia a mensagem $(\texttt{Open}, sid, P_i, P_j)$ em nome de P_i para a funcionalidade ideal $\mathscr{F}_{\mathbf{BC}}$.

   Se P_i precisar realizar a abertura com um valor incorreto, $\mathscr{S}$ também enviará para a funcionalidade uma mensagem de abertura, mesmo com o valor incorreto.

6. Sempre que o simulador $\mathscr{A}$ ordenar que uma parte seja corrompida:

   - $\mathscr{S}$ corrompe esta parte no modelo ideal e toma conhecimento de todas as suas informações internas.

   Assim:

   - $\mathscr{S}$ pode adaptar qualquer possível informação de abertura sobre um comprometimento que ainda não aberto daquela parte, como no caso de uma parte honesta realizando a abertura.

   Depois disso,

   - $\mathscr{S}$ passa toda informação ajustada para $\mathscr{A}$.

Indistinguibilidade

Para mostrar que a saída gerada pelo ambiente no modelo real é indistinguível daquela gerada no modelo ideal, três situações devem ser consideradas sendo associada a cada uma delas uma variável aleatória:

Modelo Real com *string* genuino:

A saída gerada por $\mathscr{E}$ é resultado da execução no modelo real com as partes P_i executando o protocolo na presença do adversário $\mathscr{A}$. Desta forma, σ é escolhido segundo uma distribuição uniforme e pk_0, pk_1 aleatórios são obtidos executando *KGen*, sendo usados como a *string* pública; então o protocolo é executado no modelo real com $\mathscr{A}$ e $\mathscr{E}$ com esse *string*. A esta situação está associada a variável aleatória R_g.

Modelo Real com *string* falso:

A saída gerada por $\mathscr{E}$ é resultado da execução no modelo real. O *string* falso σ é escolhido juntamente com os valores aleatórios pk_0, pk_1 como no simulador, envolvendo os valores previamente selecionados de $G_{pk_0}(r_0)$ e $G_{pk_1}(r_1)$. O protocolo é executado no modelo real com $\mathscr{A}$ e $\mathscr{E}$ usando o *string* falso. Se uma parte honesta for se comprometer com um bit b, ela deve computar o comprometimento usando os valores previamente selecionados: $y = G_{pk_b}(r)$ para $b = 0$, e $y = G_{pk_b}(r) \oplus \sigma$ para $b = 1$. se mais adiante a parte honesta for realizar a abertura, ela deverá fazê-lo enviando b e r_b. Ao final da execução, a saída será que o ambiente $\mathscr{E}$ vier a gerar. A esta situação está associada a variável aleatória R_f.

Modelo Ideal com *string* falso:

A saída gerada por $\mathscr{E}$ é resultado da execução no modelo ideal com $\mathscr{S}$ e $\mathscr{F}^{\mathbf{BC}}$, usando um *string* falso escolhido por $\mathscr{S}$. A esta situação está associada a variável aleatória I_f.

A questão da indistinguibilidade se reduz a estabelecer a indistinguibilidade entre as variáveis aleatórias R_g e R_f, e entre R_f e I_f. Isto estabelecido, por transitividade temos a indistinguibilidade entre R_g e I_f. Em ambos os casos, o problema vai se reduzir a distinguir entre uma sequência aleatória e uma pseudo aleatória.

Indistinguibilidade entre R_g e R_f

Inicialmente, será assumido que o ambiente $\mathscr{E}$ distingue R_g de R_f, com probabilidade não desprezível. Como será visto mais à frente, vai se chegar a uma contradição. A partir dessa hipótese, será construído um algoritmo que decide se uma entrada é verdadeiramente aleatória ou pseudo aleatória.

Dados um parâmetro de segurança n, uma chave pública aleatória pk de um gerador *trapdoor* pseudo aleatório $G_{pk} : \{0,1\}^n \to \{0,1\}^{4n}$ juntamente com um *string* $x \in \{0,1\}^{4n}$, que tanto pode ser escolhido aleatoriamente ou ser gerado por G_{pk}. Deseja-se decidir como x foi gerado.

Para um x aleatório de um x pseudo aleatório, o ambiente $\mathscr{E}$ será usado, distinguindo entre R_g e R_f. Para isto, um *string* σ é gerado por procedimento similar ao de $\mathscr{S}$, mas usado o dado *string* x. Então, emula-se a execução no modelo real simulando o cenário em que todas as partes são honestas, lendo todas as mensagens enviadas por $\mathscr{E}$. Especificamente:

- geração do *string* público:

  - um bit c é escolhido aleatoriamente
  - faz-se $pk_{1-c} = pk$ para uma dada chave pública
    * o bit c será o palpite para o bit ao qual uma parte honesta se compromete
  - um novo par de chaves é gerado $(pk_c, td_c) \leftarrow KGen(1^n)$
  - $r_c \{0,1\}^n$ é escolhido aleatoriamente
  - faz-se $\sigma = G_{pk_c}(r_c) \oplus x$

- emulação:
  - a simulação do protocolo é iniciada com $\mathscr{A}$ e $\mathscr{E}$ usando σ, pk_0, pk_1
  - se $\mathscr{E}$ manda que uma parte não corrompida P_i se comprometa com um bit b,
    * se $b \neq c$, então a execução para imediatamente com saída 0;
    * no caso em que $b = c$, $G_{pk_c}(r_c)$ é enviado se $b = c = 0$,
    * e caso contrário, $b = c = 1$, x é enviado em nome de P_i e a simulação continua;
  - quando mais tarde $\mathscr{E}$ mandar P_i abrir o comprometimento
    * b (que é igual a c) e r_c são enviados
  - de forma similar,
    * no caso da parte P_i ser corrompida, b e r_c são enviados para $\mathscr{A}$ antes da abertura
  - se o adversário $\mathscr{A}$ corrompeu a parte P_i antes desta se comprometer
    * a execução para com probabilidade $1/2$ (tem-se assim simetria com o caso anterior, simplificando a análise);
  - caso contrário,
    * a simulação continua.

- saída:
  - se a execução ainda não parou, a saída de $\mathscr{E}$ é copiada.

Para analisar a vantagem do algoritmo acima, considere-se primeiro o caso em que x é um *string* de $4n$ bits, uniformemente distribuído. Então σ também é aleatório e a predição c não passa nenhuma informação para $\mathscr{A}$ e $\mathscr{E}$ no início da execução. Assim, a probabilidade de uma parada prematura com saída 0 é $1/2$, independente de $\mathscr{A}$ estar dominando o comprometimento, ou deste ser efetuado por uma parte honesta. Condicionada a terminação a se dar no último estágio, a saída de $\mathscr{E}$ tem distribuição idêntica à de R_g.

Seja x agora gerado por G_{pk_c}. Nesse caso, σ corresponde a um *string* falso. Aqui também, o *string* público não revela nada sobre c para $\mathscr{A}$ e $\mathscr{E}$. Conclui-se novamente que a parada prematura ocorre com probabilidade $1/2$, independente de quem realiza o comprometimento. Além disso, dado que se atinge o estágio final, a saída de $\mathscr{E}$ tem distribuição idêntica à de R_f.

Consequentemente, em ambos os experimentos, R_g e R_f, a saída é 1 com metade da probabilidade de $\mathscr{E}$ retornar 1. Segue que se a vantagem de $\mathscr{E}$ distinguir entre R_g e R_f é dada por (onde $\mathtt{Out}(\mathscr{E},X)$ é o valor da saída gerada por $\mathscr{E}$ no experimento X):

$$\varepsilon(n) = \left| Prob\left[\mathtt{Out}(\mathscr{E},R_g) = 1\right] - Prob\left[\mathtt{Out}(\mathscr{E},R_f) = 1\right]\right|$$

assim a vantagem em distinguir entre entradas aleatórias e pseudo aleatórias é igual a $\varepsilon(n)/2$. Se $\varepsilon(n)$ não for desprezível, $\varepsilon(n)/2$ também não é. Entretanto, isto contradiz a pseudo aleatoriedade do gerador.

Indistinguibilidade entre R_f e I_f

Excetuando-se o caso em que o adversário envia algum y^* na imagem de G_{pk_0} para depois abrir o comprometimento com $b^* = 1$, os dois experimentos são idênticos. Assim, é suficiente estimar a probabilidade deste erro acontecer. Como se verá, esta probabilidade é desprezível, dada a pseudo aleatoriedade dos geradores.

Suponha que no experimento R_f a probabilidade de $\mathscr{A}$ comprometer uma parte corrompida usando y^*, tal que y^* e $y^* \oplus \sigma$ estão nas imagens de G_{pk_0} e G_{pk_1}, respectivamente, seja não desprezível. Constrói-se o algoritmo que segue.

As entradas são n, uma chave pública pk, e um *string* x de $4n$ bits. A saída é um bit indicando se x é aleatório ou pseudo aleatório. O algoritmo especifica:

1. faça:
   - $pk_1 = pk$
   - gere outro par de chaves (pk_0, td_0)
   - defina $\sigma = G_{pk_0}(r_0) \oplus x$ para $r_0 \in \{0,1\}^n$
2. emule o experimento I_f com $\mathscr{S},\mathscr{E}$ usando σ, pk_0, pk_1
   - se uma parte honesta for instruída a se comprometer, aborte
3. Se $\mathscr{A}$ instrui uma parte corrompida a se comprometer usando y^*,
   - verificar, usando td_0, se y^* está na imagem de G_{pk_0}
   - se a parte corrompida também faz uma abertura correta de y^* para $b = 1$
     - pare com saída 1.
4. para todos os outros casos,
   - retorne 0.

Observe que o algoritmo retorna 1 se a verificação com td_0 leva a um r_0^* na imagem de G_{pk_0} e se o adversário também revela r_1^* tal que

$$G_{pk_0}(r_0^*) = y^* = G_{pk_1}(r_1^*) \oplus \sigma = G_{pk_1}(r_1^*) \oplus G_{pk_0}(r_0) \oplus x$$

mas para x aleatório a probabilidade de

$$x \in \left\{ G_{pk_0}(r_0) \oplus G_{pk_0}(r_0^*) \oplus G_{pk_1}(r_1^*) |, r_0, r_0^*, r_1^* \in \{0,1\}^n \right\}$$

é no máximo 2^{-n}. Assim, nesse caso, o algoritmo dá saída 1 apenas com probabilidade exponencialmente pequena. Por outro lado, se x é pseudo aleatório então o algoritmo retorna 1 com a mesma probabilidade que o adversário $\mathscr{A}$ incorre na situação descrita para o experimento I_f. Por hipótese, essa probabilidade é não desprezível. Logo, a vantagem geral do algoritmo é não desprezível também, refutando a pseudo aleatoriedade do gerador.

Assim, a prova está concluída. □

4.7. Conclusão

O *framework* UC, conforme apresentado, é uma ferramenta poderosa para o projeto e análise de protocolos criptográficos. O framework proporciona o ferramental para a representação dos protocolos e dos seu requisitos de segurança específicos, relacionados com as tarefas que se pretendem realizar. O conceito de segurança UC, aqui expresso como implementação segura captura a motivação principal de viabilizar o projeto modular de protocolos. O Teorema da Composição sumariza todo o esforço de formalização do *framework*, de certa forma validando a definição de segurança UC. Sua prova, apesar de não ter sido apresentada, serve como um *template* para a prova de segurança UC de protocolos específicos.

Como vimos o *framework* UC inicia sua construção modelando o conceito de protocolo a partir de ITMs, evoluindo para sistemas de ITMs e definindo um modelo de execução. Nesse modelo, além se contemplar a modelagem das chamadas de subrotina, ponto básico para o tratamento da composição, tem-se juntamente com a execução do protocolo, a sua interação com o ambiente e com o adversário.

Para estabelecer o Teorema da Composição são definidos dois refinamentos ao modelo de execução, que incluem as regras de trocas de mensagens entres as partes. Estes refinamentos levam aos modelos real e ideal, que essencialmente diferem quanto ao acesso que o adversário tem aos canais de comunicação. Além disso, a separação entre a funcionalidade ideal, que encapsula de forma abstrata a tarefa a ser realizada, e o protocolo real, que pretende implementar concretamente a funcionalidade desejada, é mais um ponto esclarecedor no tratamento dado ao projeto de protocolos.

Para exemplificação da utilização do framework, que foi o foco principal desta abordagem, foi escolhido o comprometimento de bit. O conceito de comprometimento foi utilizado para demonstrar duas aplicações do *framework* UC. Inicialmente, descrito como uma funcionalidade ideal, foi demonstrado que como definida, não é possível implementar seguramente (UC segura) a funcionalidade sem que se recorra a hipóteses adicionais. Na sequência, foi construído um protocolo de comprometimento UC seguro que inclui uma hipótese de setup. Tanto a impossibilidade da implementação da funcionali-

dade ideal *pura* quanto a segurança UC do protocolo *real* ilustram o poder do *framework*, que como já indicado tem sido usado no contexto mais amplo de computação segura multipartes

4.8. Referências

Referências

[Barak et al. 2001] Barak, B., Goldreich, O., Goldwasser, S., and Lindell, Y. (2001). Resettably-sound zero-knowledge and its applications. In *APPEARED IN 42ND FOCS*, pages 116–125. IEEE Computer Society Press.

[Beaver 1991] Beaver, D. (1991). Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4:75–122.

[Beaver 1996] Beaver, D. (1996). Adaptive zero knowledge and computational equivocation (extended abstract). In *28th Annual ACM Symposium on Theory of Computing*, pages 629–638, Philadephia, Pennsylvania, USA. ACM Press.

[Beaver and Haber 1992] Beaver, D. and Haber, S. (1992). Cryptographic protocols provably secure against dynamic adversaries. In Rueppel, R. A., editor, *Advances in Cryptology – EUROCRYPT'92*, volume 658 of *Lecture Notes in Computer Science*, pages 307–323, Balatonfüred, Hungary. Springer, Berlin, Germany.

[Bellare et al. 1998a] Bellare, M., Canetti, R., and Krawczyk, H. (1998a). A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *30th Annual ACM Symposium on Theory of Computing*, pages 419–428, Dallas, Texas, USA. ACM Press.

[Bellare et al. 1998b] Bellare, M., Pointcheval, D., and Rogaway, P. (1998b). Relations among notions of security for public-key encryption schemes. In *CRYPTO ´98*, pages 26–45. Springer-Verlag.

[Bellare and Rogaway 1994] Bellare, M. and Rogaway, P. (1994). Entity authentication and key distribution.

[Ben-Or et al. 1993] Ben-Or, M., Canetti, R., and Goldreich, O. (1993). Asynchronous secure computation. In *25th Annual ACM Symposium on Theory of Computing*, pages 52–61, San Diego, California, USA. ACM Press.

[Blum 1982] Blum, M. (1982). Coin flipping by telephone. pages 133–137.

[Blum and Micali 1984] Blum, M. and Micali, S. (1984). How to generate cryptographically strong sequences of pseudorandom bits. *SIAM Journal on Computing*, 13(4):850–864.

[Brassard et al. 1988] Brassard, G., Chaum, D., and Crépeau, C. (1988). Minimum disclosure proofs of knowledge. *J. Comput. Syst. Sci.*, pages 156–189.

[Canetti 1998] Canetti, R. (1998). Security and composition of multi-party cryptographic protocols. *JOURNAL OF CRYPTOLOGY*, 13:2000.

[Canetti 2000] Canetti, R. (2000). Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067. revised Jan 2005 and Dec 2005.

[Canetti 2001] Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, USA. IEEE Computer Society Press.

[Canetti 2003] Canetti, R. (2003). Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239. `http://eprint.iacr.org/`.

[Canetti et al. 2010] Canetti, R., Chari, S., Halevi, S., Pfitzmann, B., Roy, A., Steiner, M., and Venema, W. (2010). Composable security analysis of os services. Cryptology ePrint Archive, Report 2010/213. `http://eprint.iacr.org/`.

[Canetti et al. 1996] Canetti, R., Feige, U., Goldreich, O., and Naor, M. (1996). Adaptively secure multi-party computation. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, STOC '96, pages 639–648, New York, NY, USA. ACM.

[Canetti and Gajek 2010] Canetti, R. and Gajek, S. (2010). Universally composable symbolic analysis of diffie-hellman based key exchange. Cryptology ePrint Archive, Report 2010/303. `http://eprint.iacr.org/`.

[Canetti et al. 1999] Canetti, R., Goldreich, O., Goldwasser, S., and Micali, S. (1999). Resettable zero-knowledge. In *In 32nd STOC*, pages 235–244.

[Canetti et al. 2005] Canetti, R., Halevi, S., Katz, J., Lindell, Y., and MacKenzie, P. (2005). Universally composable password-based key exchange. Cryptology ePrint Archive, Report 2005/196. `http://eprint.iacr.org/`.

[Canetti and Herzog 2004] Canetti, R. and Herzog, J. (2004). Universally composable symbolic analysis of cryptographic protocols (the case of encryption-based mutual authentication and key exchange). Cryptology ePrint Archive, Report 2004/334. `http://eprint.iacr.org/`.

[Canetti and Krawczyk 2001] Canetti, R. and Krawczyk, H. (2001). Analysis of key-exchange protocols and their use for building secure channels. In Pfitzmann, B., editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 453–474, Innsbruck, Austria. Springer, Berlin, Germany.

[Chari et al. 2011] Chari, S., Jutla, C., and Roy, A. (2011). Universally composable security analysis of oauth v2.0. Cryptology ePrint Archive, Report 2011/526. `http://eprint.iacr.org/`.

[Damgård 1990] Damgård, I. (1990). On the existence of bit commitment schemes and zero-knowledge proofs. In Brassard, G., editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 17–27, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Di Crescenzo et al. 1998] Di Crescenzo, G., Ishai, Y., and Ostrovsky, R. (1998). Non-interactive and non-malleable commitment. In *30th Annual ACM Symposium on Theory of Computing*, pages 141–150, Dallas, Texas, USA. ACM Press.

[Dodis and Micali 2000] Dodis, Y. and Micali, S. (2000). Parallel reducibility for information-theoretically secure computation. In Bellare, M., editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 74–92, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Dolev et al. 2000] Dolev, D., Dwork, C., and Naor, M. (2000). Nonmalleable cryptography. *SIAM J. Comput.*, 30:391–437.

[Dowsley et al. 2009] Dowsley, R., Müller-Quade, J., Otsuka, A., Hanaoka, G., Imai, H., and Nascimento, A. C. A. (2009). Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. Cryptology ePrint Archive, Report 2009/273. `http://eprint.iacr.org/`.

[Dowsley et al. 2010] Dowsley, R., van de Graaf, J., Marques, D., and Nascimento, A. C. A. (2010). A two-party protocol with trusted initializer for computing the inner product. Cryptology ePrint Archive, Report 2010/289. `http://eprint.iacr.org/`.

[Dwork et al. 1998] Dwork, C., Naor, M., and Sahai, A. (1998). Concurrent zero-knowledge. In *30th Annual ACM Symposium on Theory of Computing*, pages 409–418, Dallas, Texas, USA. ACM Press.

[Even et al. 1985] Even, S., Goldreich, O., and Lempel, A. (1985). A randomized protocol for signing contracts. *Commun. ACM*, 28:637–647.

[Fischlin and Fischlin 2000] Fischlin, M. and Fischlin, R. (2000). Efficient non-malleable commitment schemes. In Bellare, M., editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 413–431, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Gajek et al. 2008] Gajek, S., Manulis, M., Pereira, O., Sadeghi, A.-R., and Schwenk, J. (2008). Universally composable security analysis of tls—secure sessions with handshake and record layer protocols. Cryptology ePrint Archive, Report 2008/251. `http://eprint.iacr.org/`.

[Galil et al. 1988] Galil, Z., Haber, S., and Yung, M. (1988). Cryptographic computation: Secure faut-tolerant protocols and the public-key model. In Pomerance, C., editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 135–155, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Garay and MacKenzie 2000] Garay, J. A. and MacKenzie, P. D. (2000). Concurrent oblivious transfer. In *41st Annual Symposium on Foundations of Computer Science*, pages 314–324, Redondo Beach, California, USA. IEEE Computer Society Press.

[Goldreich 2000] Goldreich, O. (2000). *Foundations of Cryptography: Basic Tools*. Cambridge University Press, New York, NY, USA.

[Goldreich and Krawczyk 1990] Goldreich, O. and Krawczyk, H. (1990). On the composition of zero-knowledge proof systems. *SIAM Journal on Computing*, 25:169–192.

[Goldreich et al. 1987] Goldreich, O., Micali, S., and Wigderson, A. (1987). How to play any mental game, or a completeness theorem for protocols with honest majority. In Aho, A., editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City,, New York, USA. ACM Press.

[Goldreich and Oren 1994] Goldreich, O. and Oren, Y. (1994). Definitions and properties of zero-knowledge proof systems. *Journal of Cryptology*, 7(1):1–32.

[Goldwasser and Levin 1991] Goldwasser, S. and Levin, L. A. (1991). Fair computation of general functions in presence of immoral majority. In Menezes, A. J. and Vanstone, S. A., editors, *Advances in Cryptology – CRYPTO'90*, volume 537 of *Lecture Notes in Computer Science*, pages 77–93, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Goldwasser and Micali 1984] Goldwasser, S. and Micali, S. (1984). Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299.

[Goldwasser et al. 1989] Goldwasser, S., Micali, S., and Rackoff, C. (1989). The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208.

[Gorantla et al. 2009] Gorantla, M. C., Boyd, C., and Nieto, J. M. G. (2009). Universally composable contributory group key exchange. Cryptology ePrint Archive, Report 2009/300. `http://eprint.iacr.org/`.

[Green and Hohenberger 2008] Green, M. and Hohenberger, S. (2008). Universally composable adaptive oblivious transfer. Cryptology ePrint Archive, Report 2008/163. `http://eprint.iacr.org/`.

[Hirt and Maurer 2000] Hirt, M. and Maurer, U. M. (2000). Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60.

[Katz et al. 2011] Katz, J., Maurer, U., Tackmann, B., and Zikas, V. (2011). Universally composable synchronous computation. Cryptology ePrint Archive, Report 2011/310. `http://eprint.iacr.org/`.

[Kuesters and Tuengerthal 2009] Kuesters, R. and Tuengerthal, M. (2009). Universally composable symmetric encryption. Cryptology ePrint Archive, Report 2009/055. `http://eprint.iacr.org/`.

[Kurosawa and Furukawa 2008] Kurosawa, K. and Furukawa, J. (2008). Universally composable undeniable signature. Cryptology ePrint Archive, Report 2008/094. `http://eprint.iacr.org/`.

[Micali and Rogaway 1992] Micali, S. and Rogaway, P. (1992). Secure computation (abstract). In Feigenbaum, J., editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404, Santa Barbara, CA, USA. Springer, Berlin, Germany.

[Naor 1991] Naor, M. (1991). Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158.

[Naor et al. 1998] Naor, M., Ostrovsky, R., Venkatesan, R., and Yung, M. (1998). Perfect zero-knowledge arguments for np can be based on general complexity assumptions (extended abstract). *JOURNAL OF CRYPTOLOGY*, 11:87–108.

[Naor and Yung 1990] Naor, M. and Yung, M. (1990). Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, STOC '90, pages 427–437, New York, NY, USA. ACM.

[Pfitzmann et al. 2000] Pfitzmann, B., Schunter, M., and Waidner, M. (2000). Secure reactive systems.

[Pfitzmann and Waidner 1994] Pfitzmann, B. and Waidner, M. (1994). A general framework for formal notions of "secure"systems. In *SYSTEM, HILDESHEIMER INFORMATIK-BERICHTE 11/94, UNIVERSITAT*.

[Pfitzmann and Waidner 2000] Pfitzmann, B. and Waidner, M. (2000). Composition and integrity preservation of secure reactive systems. In *In Proc. 7th ACM Conference on Computer and Communications Security*, pages 245–254. ACM Press.

[Rabin 1981] Rabin, M. (1981). How to exchange secrets using oblivious transfer. Technical report, Tech. Memo TR-81, Aiken Computation Laboratory, Harvard University.

[Rackoff and Simon 1992] Rackoff, C. and Simon, D. R. (1992). Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, pages 433–444, London, UK. Springer-Verlag.

[Shoup 1999] Shoup, V. (1999). On formal models for secure key exchange. Technical report.

[Unruh and Müller-Quade 2009] Unruh, D. and Müller-Quade, J. (2009). Universally composable incoercibility. Cryptology ePrint Archive, Report 2009/520. `http://eprint.iacr.org/`.

[Yao 1982] Yao, A. C. (1982). Theory and applications of trapdoor functions. In *23rd Annual Symposium on Foundations of Computer Science*, pages 80–91, Chicago, Illinois. IEEE Computer Society Press.

A. Gerador Pseudo Aleatório

Informalmente, uma distribuição $\mathscr{D}$ é pseudo aleatória se nenhum *distinguisher*, aqui referido como diferenciador *D*, que execute em tempo polinomial consegue detectar se lhe foi passado um *string* amostrado de $\mathscr{D}$ ou se ele foi escolhido uniformemente ao acaso. Na formalização, será exigido que o algoritmo *D*, em tempo polinomial, gere a saída 1 com quase a mesma probabilidade tanto para um valor vindo de $\mathscr{D}$, quanto para uma entrada verdadeiramente aleatória. Essa saída pode ser interpretada como um *guess*.

Um gerador pseudo aleatório é um algoritmo determinístico que recebe na entrada um pequeno *string* verdadeiramente aleatório e o expande em um *string* mais longo que é pseudo aleatório. Dito de outra forma, um gerador pseudo aleatório usa uma pequena quantidade de aleatoriedade para gerar uma quantidade maior pseudo aleatória.

Na definição que segue, n é o comprimento do *string* de entrada e $l(n)$ é o comprimento da saída, sendo interessantes apenas os casos em que $l(n) > n$.

Definição A.1. *Sejam $l(.)$ um polinômio e G um algoritmo determinístico em tempo polinomial tal que, para qualquer entrada $s \in \{0,1\}^n$, o algoritmo G gera como saída um string de comprimento $l(n)$. Diz-se que G é um gerador pseudo aleatório se:*

- *para todo n, $l(n) > n$*
- *para todo diferenciador D em tempo polinomial,*

$$|Pr[D(r) = 1] - Pr[D(G(s)) = 1]|$$

é desprezível. Onde r é escolhido uniformemente ao acaso de $\{0,1\}^{l(n)}$. $l(n)$ é o fator de expansão de G.

Um diferenciador pode ser definido como um algoritmo D em tempo polinomial que recebe um *string* de entrada $x \in \{0,1\}^{l(n)}$ e gera como saída $D(x) = 1$ se x é verdadeiramente aleatório ou $D(x) = 0$ caso contrário.

Definição A.2. *Seja D um algoritmo em tempo polinomial, que recebendo um valor $x \in \{0,1\}^{l(n)}$ como entrada e correspondentemente gera como saída $D(x)$ tal que:*

$$D(x) = \begin{cases} 1, & \textit{se x é verdadeiramente aleatório} \\ 0, & \textit{caso contrário} \end{cases}$$