Security and Cooperation in Wireless Networks

Additional Problems

edited by
Levente Buttyán and Jean-Pierre Hubaux

July 2009
Lausanne

Preface

The problems hereafter have been generated by students participating in the course *Security and Co-operation in Wireless Networks*. The course is part of the doctoral school of EPFL in information and communication sciences (http://phd.epfl.ch/edic) and its URL is http://secowinetcourse.epfl.ch/.

We are particularly grateful to students Joppe Bos, Zarko Milosevic, Seyyd Hasan Mirjalili, and Onur Özen, whose contributions were convincing enough to be included in the present document.

Problems

Problem 1

In self-organized mobile networks, there is the need for nodes to be able to generate their own addresses and verify the ones from others. One technique to solve this problem is using self-certifying addresses, which allows hosts and domains to prove that they have the address they claim to have without relying on any global trusted authority. The notion of a self-certifying name is straightforward: the name of the object is the public-key (or, for convenience the hash of the public-key) that corresponds to that object (similar to the concept of CGA as described in Chapter 4 of the book).

(a) Compare the efficiency/security trade-off between address generation with hashing the public-key and without hashing (i.e., by using the public-key itself)? What are the advantages and disadvantages of both techniques?
(b) Recall the basic security properties of a cryptographic hash function. What are the computational complexities of the brute force attacks aiming to defeat those properties?

Consider CGA (described in Chapter 4 of the book) without hash extensions (i.e., when parameter $sec = 0$) where the addresses are generated by only hashing the public-key, subnet prefix and collision count.

(c) How do the generic attacks on the underlying hash function relate to CGA without hash extension? What are the computational complexities of these generic attacks on CGA without hash extension?

Now consider CGA with hash extension:

(d) Calculate the number of SHA-1 evaluations needed to generate an IPv6 address using the value of sec as a parameter. What is the security/efficiency trade-off to generate an IPv6 address by increasing values of sec?
(e) What are the computational complexities of the generic attacks on CGA with hash extension?

Problem 2

Observe that in CGA (described in Chapter 4 of the book) the subnet prefix is not used in the computation of Hash2.

(a) How can this observation be used to perform an attack? (*Hint:* Use a time-memory trade-off.)
(b) What is the overall complexity of this attack (required storage, time)?
(c) Does including the subnet prefix in the computation of Hash2 prevents this attack? What is the disadvantage of using the subnet prefix here?

Problem 3

Let us consider the following trust estimation engine for a server. The basic idea is to rate users after the completion of some transactions to derive a trust score, which can assist the system in deciding whether or not to transact with that user in the future. In fact, the system attempts to measure the trustworthiness of a user.

Suppose the server's rating of each transaction is binary, i.e., positive or negative. Posteriori probabilities of binary events can be represented as Beta distribution [1].

[1] In probability theory and statistics, the beta distribution is a family of continuous probability distributions defined on

We can interpret trust as the probability expectation with which positive behavior will happen in the future. The probability expectation value of the Beta distribution is given as:

$$E(p) = \frac{\alpha}{\alpha + \beta}$$

after observing $\alpha - 1$ independent events with probability p and $\beta - 1$ with probability $1 - p$, if the prior distribution of p was uniform.

Let *Trust Score* of user i, denoted by Θ_i, be equal to $E(p)$. Furthermore, let r be the observed number of positive outcomes and s be the observed number of negative outcomes.

(a) What is the *Trust Score* of user i after T transactions? Assume that at the beginning, when the server does not have any experience with the user (i.e. $r, s = 0$), $\Theta_i(0) = \frac{1}{2}$, i.e. a neutral opinion about the user. Note that $0 < \Theta_i(T) < 1$ for all T and $\Theta_i(T) \approx 0$ means distrust and $\Theta_i(T) \approx 1$ means trust.
(b) Old behavior may not always be relevant for the actual trust score, because the user may change its behavior over time. What is needed is a model which gives less weight to old behaviors and more weight to recent ones. This translates into gradually forgetting old behavior. Introduce a forgetting factor $\lambda \in [0, 1]$ in your equation which can be adjusted according to the expected rapidity of change in the observed user. Assuming $\lambda = 1$ means nothing is forgotten. The other extreme is when $\lambda = 0$ which means only the last behavior rating to be counted and all others to be completely forgotten. Here the order in which rating was given is important.
(c) The equations in Parts (a) and (b) can be written in recursive way, i.e. $\Theta_i(t) = f(\Theta_i(t-1))$. If you have written the equations in non-recursive way, the disadvantage is that all ratings given by the system should be kept. This can be avoided by transferring your equation to a recursive equation. Translate your equation in Part (b) to a recursive function.

Problem 4

In Sections 8.3.1 and 8.3.2 of the book, it is shown that coordinated changing of pseudonyms inside mix zones is one possible solution for providing location privacy. This solution assumes that all nodes change pseudonyms inside mix zones (nodes are always cooperative) which may not be a realistic assumption as changing a pseudonym has a cost (consisting of obtaining the new pseudonym, routing overhead due to changing the pseudonym, *etc.*). The goal of this exercise is to model, using game theory (See Appendix B), the pseudonym changing approach for achieving location privacy under the assumption that nodes are not always cooperative (rather the nodes are rational).

(a) Define a strategic-form game that represents the pseudonym changing approach (let's call that the pseudonym changing game) assuming that

- two players meets in a mix zone and engage in the game;
- the players have two possible strategies: C - changing pseudonym (or cooperating) and D - no pseudonym change (defecting);
- the achieved level of privacy L is equal to $\log_2(n)$ where n is the number of players that changed pseudonym (i.e., played C); if $n = 0$ the achieved level of privacy is equal to 0;

the interval $[0, 1]$ parameterized by two positive shape parameters, typically denoted by α and β. The beta distribution is the conjugate prior of the binomial distribution.

- the cost of changing the pseudonym is γ; and
- the goal of each player is to maximize its utility (the level of its privacy).

(b) Identify the Nash equilibria (NE). What is the Pareto-optimal NE strategy profile?
(c) Let us modify the pseudonym changing game such that the player P_2 is malicious, i.e., the goal of player P_2 is to minimize the utility of the rational player P_1. The gain $G(P_2)$ of P_2 is defined as $G(P_2) = 1 - L(P_1)$. The cost of changing pseudonym is γ for both players. The goal of each player is to maximize its utility, defined as the difference between the obtained gain and the incurred cost. Give the strategic-form representation of this game and identify the Nash equilibria.
(d) Let us now assume that the players can be malicious with some predefined probability q. Furthermore, let us assume that the players make their moves sequentially (i.e., the game is dynamic, see Appendix B). Player P_1 moves first and then player P_2 moves. The advantage of P_2 is that it can observe the move of player P_1. Identify the Nash equilibria in this game.

Problem 5

In the improved anonymous routing protocol that is described in Section 8.4 of the book, we introduced a counter c_{SD} whose value is synchronously maintained by the source and the destination. Does this protocol ensure forward secrecy? If so, why? If not, could the protocol be modified to ensure it?

Problem 6

This problem is related to the ElGamal asymmetric-key encryption scheme.

(a) Assume Alice and Bob use the ElGamal asymmetric-key encryption scheme without the use of certificates; i.e. without ensuring the authenticity of the public keys. Think of a way for Eve to successfully read and possibly modify messages going from Alice to Bob without either of them noticing.
(b) Show that the ElGamal scheme is unconditionally malleable, and hence it is not secure under a chosen ciphertext attack; i.e. given an encryption (R, C) of some (possibly unknown) message m, construct a valid encryption $(R', C') \neq (R, C)$ of some other message $m' \neq m$.
(c) Show that the version of ElGamal as presented in Appendix A of the book does not have the IND-CPA property (i.e., indistinguishability under chosen-plaintext attack). This means that a challenger can freely choose two messages m_0 and m_1, next he challenges someone to encrypt one of these messages and receives this encrypted message back, and he can always tell which message was encrypted. (*Hint:* What is the order of $\mathbb{Z}_p^*$?)
(d) Find a way to solve the problem stated in Part (c).

Solutions

Solution of Problem 1

Part (a)

Advantage of using the public key as the name of the object:

- *Security:* An attacker needs to break the underlying public key cryptography in order to impersonate the object since every object has an unique identity.

Disadvantage of using the public key as the name of the object:

- *Impractical:* In practice, the key sizes used in public key cryptography are much longer compared to the space reserved for the addresses in, for instance, IPv6.

Advantage of using the hash of the public key as the name of the object:

- *Practical:* Using a cryptographic hash function one can create the "fingerprint" of the public key and use this as the name of the object. This truncated fingerprint can be used inside networking protocols as the address of a node.

Disadvantages of using the hash of the public key as the name of the object:

- *Security:* The security of self-certifying addresses relies on the public key cryptography protocols and / or the cryptographic hash function used. As one introduces more cryptographic components, the whole system is as secure as the weakest component.
- *Security in practice:* The advantage of being usable in practice results in using the truncated output of a cryptographic hash function. This means the security is less compared to the level of security given by using the full hash output.

Part (b)

The basic security properties of a cryptographic hash function are

- *Collision resistance:* For an adversary, it should be *hard* to find two distinct messages M and M' such that $H(M) = H(M')$.
- *Preimage resistance:* For an adversary, given the target hash value D, it should be *hard* to find a preimage M such that $H(M) = D$.
- *Second-preimage resistance:* For an adversary, given a message M, it should be *hard* to find another different message M' such that $H(M) = H(M')$.

Complexities of generic attacks to these properties:

- Finding a collision on d bits has complexity $O(2^{\frac{d}{2}})$.
- Finding a pre-image on d bits has complexity $O(2^d)$.
- Finding a second pre-image on $d - l$ bits for any message shorter than 2^l bits has complexity $O(2^{d-l})$. This result is valid mainly for iterative constructions. For the general case, the security requirements of pre-image and the second pre-image resistances are considered to be same.

Part (c)

Collision resistance corresponds to the case where an adversary can add two nodes with the same address (but different public keys) to the network.

Computational complexity of this attack follows from the birthday attack. The attacker has to perform $O(2^{30.5})$ (the sec value is not encoded any more in the interface identifier and the u, v bits still exist) hash function evaluations to find a collision in the interface identifier (i.e. Hash1). Each evaluation requires generating valid public/private-key pairs since the modifier value is not used when $sec = 0$. Assume generating a valid public/private-key pair requires $O(2^t)$ operations in terms of hash evaluations, then the collision attack requires $O(2^{30.5+t})$ hash function evaluations in total for creating two valid nodes with the same address. The birthday attack applied to this scenario is the attack which requires negligible memory.

Pre-image resistance corresponds to the case where an adversary can pick any address in the network and generate a corresponding public key. This makes no sense in this setting since the public key is public.

Second pre-image resistance corresponds to the case where an adversary can pick any address in the network and generate a node with the same address but a different public-key. This attack is the most serious attack model as the attacker is able to impersonate a node once he can find a second pre-image with a valid public-private key pair.

The computational complexity of impersonation is obtained as follows. Let the adversary A try to impersonate the node N_0 in a given network. We define the function $BCGA(K_{\text{pub}}, SP, CC)$ where BCGA stands for Basic CGA, without the use of hash extensions, and K_{pub} is the public key of N_0, SP the subnet prefix and CC the collision count. In order to impersonate N_0, the adversary has to generate a new public/private-key pair $(K_{\text{pub}}, K_{\text{priv}})$ which produces the same IPv6 address. Here, we implicitly assume that the impersonation is done in the same subnetwork.

Before proceeding, the adversary has access to the IPv6 address and the B-CGA parameters of the attacked node N_0 as they are all public values. In order to generate the same address, the adversary A has to find a second pre-image for the 61-bit interface identifier which requires $O(2^{61})$ (the sec value is not encoded any more in the interface identifier and the u, v bits still exist) hash computations with a standard implementation and under the assumption that the underlying hash function is ideal, i.e the only attack model is the generic brute force. Thus, the second pre-image attack is done by generating random public/private-key pairs $(K_{\text{pub}}, K_{\text{priv}})$ and checking the corresponding truncated hash output. As we assume that the underlying hash function is ideal, the adversary is expected to find a second pre-image of the interface identifier with non-negligible probability after $O(2^{61})$ trials. As generating a valid public/private-key pairs requires $O(2^t)$ operations in terms of hash evaluations, the second pre-image attack requires $O(2^{61+t})$ hash function evaluations in total for impersonation.

Part (d)

Using parameter $sec = s$ requires that $16 \times s$ least significant bits of Hash2 are zero. So, one expects to make 2^{16s} SHA-1 evaluations before finding a hash value with this desired property. Increasing s by one would make it 2^{16} times harder for the node and the attacker to find the desired hash value. Hence, the security is increased at the cost of efficiency.

Part (e)

Collision attack follows from the birthday attack and is very similar to the attack done in Part (c) for CGA without hash extension. The only difference here is the effect of the modifier. In the simplest attack model, the attacker has to perform $O(2^{29.5})$ hash function evaluations to find a collision in the interface identifier which requires the generation of a valid modifier value for each evaluation. As the probability of having two valid random modifier values is 2^{-32s}, the collision attack requires $O(2^{29.5+32s})$ hash function evaluations in total for creating two valid nodes with the same address for CGA with hash extension.

The computational complexity of impersonation is obtained as follows. Let the adversary A try to impersonate the node N_0 in a given network. We define the function $CGA(m, SP, CC, K_{\text{pub}})$ where m is the modifier, SP the subnet-prefix, CC the collision count and K_{pub} the public key. Now, let the node N_0 to be impersonated and assume the function $CGA(m_0, SP, CC_0, K_{\text{pub}_0})$ with respective CGA parameters. In order to impersonate N_0, the adversary has to generate a new public/private-key pair $(K_{\text{pub}}, K_{\text{priv}})$ together with a valid modifier m, subnet prefix SP and a specific collision count CC which produces the same IPv6 address. Here, we implicitly assume that the impersonation is done in the same subnetwork.

Before proceeding, the adversary has access to the IPv6 address, and hereby also to the security parameter s, and the CGA parameters of the attacked node N_0 as they are all public values. In order to generate the same address, the adversary A has to find a second pre-image for the 59-bit digest Hash1 which requires $O(2^{59})$ computations with a standard implementation and under the assumption that the underlying hash function is ideal, i.e. the only attack model is the generic brute force. Here, the subnet prefix value is fixed to SP and collision count CC is taken to be one. Thus, the second pre-image attack to Hash1 is done by generating random public/private-key pairs $(K_{\text{pub}}, K_{\text{priv}})$ (at least one pair) and/or modifier values m where the latter is the cheapest solution. As we assume that the underlying hash function is ideal the adversary is expected to find a second pre-image $(m, SP, CC_0, K_{\text{pub}})$ of Hash1 with non-negligible probability after $O(2^{59})$ trials.

After finding a second pre-image to Hash1, the adversary has to satisfy the constraints of the hash extension. More precisely, the generated modifier m and the public key K_{pub} are hashed by Hash2 together with $64 + 8 = 72$ zero bits to construct Hash2 which is expected to satisfy $16 \times s$ zero bits in the most significant $16 \times s$ bits. Since the modifier m and the public key K_{pub} are generated randomly, the probability of having $16 \times s$ zero bits in the most significant $16 \times s$ bits of Hash2 is 2^{-16s}. Therefore, the second-preimage attack has to be mounted 2^{16s} times to satisfy the constraints. This leads to $O(2^{59+16s})$ computations in total for impersonation.

Solution of Problem 2

Part (a)

An attacker, or a legitimate node, can create a look-up table with different modifier values such that these modifier values, together with a public-key, have the desired properties for the given $sec = s$ parameter. This look-up table, or database, is independent of the subnet prefix; hence, it has to be created only once and can be used in all settings in the future. For a legitimate node, this solves the problems as discussed in 1(d) since it can look-up a new modifier value whenever it wants to renew its address. Unfortunately, it creates opportunity for an attacker as well; he can utilize this same look-up table trying to impersonate a random node in the network. An attacker needs to have a huge database of modifiers and looks for a modifier value which, together with his own (different from the legitimate node) public key will give the same Hash1 value as this legitimate node. If such a value has been found the attacker has successfully impersonated the address of this node.

Part (b)

Let us outline a procedure an attacker could follow in order to mount such an attack. Given a number of $k > 0$ networks each of size approximately 2^{n_i}, for $0 < i \leq k$, assume an attacker needs at most x calls to the hash function and comparisons of the hash-values in order to impersonate one of 2^{n_i} nodes. First of all, the attacker creates a valid public/private-key pair $(K_{\text{pub}}, K_{\text{priv}})$ once. Assume a database is given with valid modifier values m_j, $1 \leq j \leq x$, $j \in \mathbb{Z}$, such that the most significant $16 \times s$ bits of Hash2 are zero; the condition on this hash value is satisfied. In order to impersonate one of the 2^{n_i} nodes in the network the condition on the Hash1 value should be satisfied as well. In order to create the interface identifier, 59 bits from this Hash1 are used. Since the probability of finding a second pre-image is $\frac{1}{2^{59-n_i}}$, it follows that a second pre-image is expected after x hash evaluations, where $x = 2^{59-n_i}$. The cost C, the number of calls to our hash function, for creating the database of modifiers depends on the parameter s:

$$C = x \cdot 2^{16s} = 2^{59+16s-n_i}.$$

The database is independent of the currently used subnet prefix and can be computed once and used for all subsequent attacks in the future. The total cost T, for A attacks (not restricted to a specific domain) becomes

$$T = 2^{59-n_i} + \frac{2^{59+16s-n_i}}{A},$$

Asymptotically, when the number of attacks go to infinity and selecting the smallest network size among n_i which maximizes the attack cost, this becomes

$$x \leq 2^{59-\min(n_i)}$$

The storage cost is $128 \cdot 2^{59-\min(n_i)}$ bits which corresponds to $2^{33-\min(n_i)}$ Gbyte.

Part (c)

Including the subnet prefix does prevent this type of attack since the look-up table can only be used in one domain. Including the subnet prefix in the computation of the Hash2 value comes with a cost in terms of efficiency. When the node is in a mobile network and travels from domain to domain it needs to recompute the value Hash2 again when the subnet prefix changes which reduces the efficiency.

Solution of Problem 3

Part (a)

Referring to the definition in the problem:

$$\Theta_i(T) = E(p) = \frac{\alpha}{\alpha + \beta}$$

If r is the observed number of positive outcomes until time T, then $\alpha - 1 = r$ or $\alpha = r + 1$. Similarly, $\beta = s + 1$. Therefore:

$$\Theta_i(T) = \frac{r+1}{r+s+2}$$

Let $N = s + r$ be the total number of transactions, then

$$\Theta_i(T) = \frac{r+1}{N+2}$$

For $r, s = 0$, $\Theta_i(0) = \frac{1}{2}$.

Part (b)

Assuming that the system had T transactions so far, we define

$$R_\lambda(T) = \sum_{t=1}^{T} \tilde{r}_t \lambda^{(T-t)}$$

and

$$N_\lambda(T) = \sum_{t=1}^{T} \lambda^{(T-t)}$$

Then:

$$\Theta_i(T) = \frac{R_\lambda(T) + 1}{N_\lambda(T) + 2} \qquad \lambda \in [0, 1]$$

At time $t = 0$, $R_\lambda(0) = 0$ and $N_\lambda(0) = 0$. $\tilde{r}_t \in \{0, 1\}$ is a variable which indicates whether the outcome of a transaction was positive or negative. $\tilde{r}_t = 1$ means a positive outcome at time t and $\tilde{r}_t = 0$ means a negative outcome.

Part (c)

Given that

$$R_\lambda(T) = \sum_{t=1}^{T} \tilde{r}_t \lambda^{(T-t)}$$

and

$$N_\lambda(T) = \sum_{t=1}^{T} \lambda^{(T-t)}$$

We rewrite the equations in a recursive way:

$$R_\lambda(t) = R_\lambda(t-1)\lambda + \tilde{r}_t \qquad t = 1 \dots T \qquad R_\lambda(0) = 0$$

and

$$N_\lambda(t) = N_\lambda(t-1)\lambda + 1 \qquad t = 1 \dots T \qquad N_\lambda(0) = 0$$

Solution of Problem 4

Part (a)

The matrix of the game is given in Table 1.

P_1/P_2	C	D
C	$(1-\gamma, 1-\gamma)$	$(-\gamma, 0)$
D	$(0, -\gamma)$	$(0, 0)$

Table 1: Strategic form representation of the two-player pseudonym changing game described in Problem 4(a)

Part (b)

There are two Nash equilibria (C, C) and (D, D). (C, C) is the Pareto-optimal NE strategy profile.

Part (c)

The game is given in Table 2. The only Nash equilibrium is (D, D). It can also be seen that the best strategy for the malicious player is to play D and it does not depend on the move of the rational player.

P_1/P_2	C	D
C	$(1-\gamma, -\gamma)$	$(-\gamma, 1)$
D	$(0, 1-\gamma)$	$(0, 1)$

Table 2: The modified version of the pseudonym changing game in which player P_2 is malicious as described in Problem 4(c)

Part (d)

In this version of the pseudonym changing game, when the rational player P_1 meets the other player P_2 in a mix zone, P_1 has only probabilistic information about type of P_2 (which can be rational or malicious). In this case the game is solved in the following way. Player P_1 calculates the expected utility by playing C (denoted by $E(C)$) and expected utility by playing D (denoted by $E(D)$). If $E(C) > E(D)$, player P_1 plays C, otherwise it plays D.

If player P_2 is a rational player, it will observe the move of P_1, and then plays its best response. If the observed move is C, P_2 will play C, otherwise it will play D.

If P_2 is malicious, it will always play D (see Part (c)).

The expected utility of the rational player P_1 is calculated as follows:

$$E(C) = q(-\gamma) + (1-q)(1-\gamma) = 1 - \gamma - q$$

$$E(D) = 0$$

The first part of the expression for $E(C)$ corresponds to the achieved utility of player P_1 when player P_2 is malicious and the second part corresponds to the achieved utility of player P_1 when P_2 is rational. From these two expressions, it follows that the rational player P_1 will decide to change its pseudonym if $q < 1 - \gamma$, or defects otherwise.

Thus, the solution of this game is the following:

- A rational player that moves first plays C if $q < 1 - \gamma$, and plays D otherwise.
- A rational player that moves second plays C if the first player played C, and plays D otherwise.
- A malicious player always plays D.

Solution of Problem 5

No, the protocol does not provide forward secrecy. If a node is broken and its secret key k_{SD} is compromised, the adversary learns the actual counter value c_{SD} and he can compute all the past hints $h(k_{SD}, c_{SD})$ to recognize them in all past route requests (that have been recorded by the adversary), thus, the privacy of past communications is compromised.

To add forward secrecy to the protocol, one would need to use a state value instead of a counter. The source and the destination start with a state s_1 and update the state in the node using a one-way function, i.e., $s_{i+1} = H(s_i)$ where H is a one-way function.

Solution of Problem 6

Part (a)

Since there is no way of ensuring authenticity, Eve can "sit" in the middle of the conversation between Alice and Bob. Eve pretends to be Bob towards Alice and pretends to be Alice towards Bob. In this scenario Alice is using the public-key from Eve, thinking it belongs to Bob, and sends her message encrypted with this key. Eve is now able to decrypt and hereby read the message and after some modifications she can encrypt it with Bob's public-key in order to send it to Bob. Bob believes the message comes from Alice since there is no way to ensure authenticity. This attack scenario is called a "man-in-the-middle attack".

Part (b)

Given a valid encryption $(R, C) = (g^r, m \cdot g^{ar})$ one can pick a random integer $i \in \mathbb{Z}_{>0}$ and compute $(R, iC) = (g^r, i \cdot m \cdot g^{ar})$. When decrypting one obtains the valid message

$$\frac{iC}{R^a} = \frac{i \cdot m \cdot g^{ar}}{g^{ar}} = i \cdot m$$

Part (c)

One is able to do this because the order of the multiplicative group Z_p^* is $p - 1$. This order is even assuming p is prime and $p > 2$. Consider the following two definitions:

Definition 1. *For co-prime integers a and m, where m is positive, we say that a is a quadratic residue (mod m) if and only if the congruence*

$$x^2 \equiv a \pmod{m}$$

is solvable for some integer x. If the congruence is not solvable, a is said to be a quadratic non-residue (mod m).

Definition 2. *The Legendre symbol $\left(\frac{a}{p}\right)$, where p is an odd prime, is defined as*

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & \textit{if } a \equiv 0 \textit{ (mod } p\textit{)} \\ 1, & \textit{if } a \textit{ is a quadratic residue (mod } p\textit{)} \\ -1, & \textit{if } a \textit{ is a quadratic non-residue (mod } p\textit{)} \end{cases}$$

One of the useful properties of the Legendre symbol is the following:

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right).$$

In other words, the product of two residues or non-residues is a residue, whereas the product of a residue with a non-residue is a non-residue.

The idea is to create two messages m_0 and m_1 such that $\left(\frac{m_0}{p}\right) = 1$ and $\left(\frac{m_1}{p}\right) = -1$. These are the two messages the challenger hands out. Next it receives the ciphertext $(R, C) = (g^r, m_i \cdot g^{ar})$ where $i \in \{0, 1\}$. Given the generator $\langle g \rangle = \mathbb{Z}_p^*$, $A = g^a$ (which is part of the public key) and g^r (which is part of the ciphertext) he responds in the following way:

```
if $\left(\frac{g^a}{p}\right) = \left(\frac{g^r}{p}\right) = -1$ then
    if $\left(\frac{C}{p}\right) = 1$ then respond $m_1$
    else respond $m_0$
else
    if $\left(\frac{C}{p}\right) = 1$ then respond $m_0$
    else respond $m_1$
```

Lets see why the response is correct. If $\left(\frac{g^a}{p}\right) = \left(\frac{g^r}{p}\right) = -1$ (the first case) then $\left(\frac{g^a}{p}\right)\left(\frac{g^r}{p}\right) = \left(\frac{g^{ar}}{p}\right) = (-1)^2 = 1$. Since we received $C = m_i \cdot g^{ar}$, we can compute the Legendre symbol of this value and this immediately tells us which message was encrypted because the signs of the Legendre symbol of m_0 and m_1 differ. One can use exactly the same strategy for the other case.

Part (d)

A possible solution is to work in prime order subgroups of $\mathbb{Z}_p^*$. One way of achieving this is by using the following theorem:

Theorem 1. *Let p be an odd prime. Then* $|(\mathbb{Z}_p^*)^2| = \frac{p-1}{2}$.

This means that $(\mathbb{Z}_p^*)^2$ is a subgroup of $\mathbb{Z}_p^*$ and half of the elements of $\mathbb{Z}_p^*$ are quadratic residues. If we choose our prime p as $p = 2q + 1$ where q is prime then the order of an element $x \in Z_p^*$ with $x \not\equiv \pm 1 \bmod p$ is $\frac{p-1}{2} = q$: a prime number.