Fraud within Asymmetric Multi-Hop Cellular Networks

Gildas Avoine

EPFL

Lausanne, Switzerland

Abstract. At *Financial Cryptography 2003*, Jakobsson, Hubaux, and Buttyán suggested a lightweight micro-payment scheme aimed at encouraging routing collaboration in asymmetric multi-hop cellular networks. We will show in this paper that this scheme suffers from some weaknesses. Firstly, we will describe an attack which enables two adversaries in the same cell to communicate freely without being challenged by the operator center. We will put forward a solution to fix this protocol. Then we will describe another method that allows an attacker to determine the secret keys of the other users. This attack thwarts the micro-payment scheme's purpose because an attacker can thus communicate without being charged. Finally we will suggest some solutions to counteract this attack.

Key words: micro-payment, multi-hop cellular networks, cryptanalysis.

1 Introduction

Nowadays, architectures for wireless communication are mostly based on *single-hop cellular networks*, e.g. the Global System for Mobile communications (GSM) [1]. Within this framework, mobile stations can access the infrastructure with a single hop and base stations can also reach each mobile station in its cell with one hop. However, such infrastructures require multiple fixed base stations to encompass the service area, which can lead to numerous problems. Conversely, *Multi-hop networks*, also called *ad-hoc networks*, do not rely on a fixed infrastructure; mobile stations communicate amongst themselves using multi-hop routing. Though these networks have some advantages, mainly their low cost, they bring with them several problems, chiefly related to the routing process (congestion, selfishness, etc.). *Multi-hop cellular networks* [3] mitigate these problems by combining conventional single-hop cellular networks and multi-hop networks. Here, the multi-hop routing is only used in order to reach the closest base station and to link the destination base station with the recipient user. A variant of this kind of network, introduced by Jakobsson, Hubaux, and Buttyán [2] consists of a multi-hop uplink, i.e., the link from the mobile station to the base station, and a single-hop downlink, i.e., the link from the base station to the mobile station. Such a network, called an *asymmetric multi-hop cellular network*, aims to reduce

the energy consumption of the mobile stations.
When a routing protocol is based on multi-hop links, incentives must be used to encourage cooperation between the parties – called *nodes* in this case. Micro-payments are one way to treat the problem. In this paper, we analyze the lightweight micro-payment scheme suggested by Jakobsson, Hubaux, and Buttyán [2] at *Financial Cryptography 2003*, which aims to encourage cooperation in asymmetric multi-hop cellular networks. In this scheme, the cost paid by the packets' originators covers on average the routing cost, which includes the (probabilistic) gain of the intermediaries along the packet route. We will show in this paper that the proposed scheme suffers from some weaknesses which compromise its security. In Section 2, we will recap the main principles of the analyzed micro-payment scheme. In Section 3, we will firstly describe a method which allows two attackers in the same cell to communicate freely; we will then suggest a lightweight patch in order to fix the scheme. In Section 4, we will describe another threat, which enables an attacker to determine the secret keys of the nodes. This attack thwarts the micro-payment scheme's purpose because, with these keys, an attacker can communicate without being charged; the owners of the stolen keys are charged instead. Finally, we will also suggest mechanisms to counteract this using keyed-hash functions.

2 Description of the scheme

2.1 Entities

The micro-payment scheme suggested by Jakobsson *et al.* [2] consists of three classes of entities: the *users*, the *base stations* and the *operator centers*. Among the users, we distinguish between the *originators* of the packets, the *intermediaries* on the path from the originator to the base station and the *recipients* of the packets. We also recognize the *base stations of the home network* of a user, i.e., the network where the user is registered and the *base stations of the foreign networks*. There is an *operator center* per network, which is simultaneously an accounting, auditing and registration center.

2.2 Principle

Before sending a packet, an originator has to send a forward request including a reward level L to his neighbors, one after another, until one of them agrees to forward the packet. The reward expected by the participating neighbor is related to L. Increasing the reward level allows users with particularly low battery power to obtain service in a neighborhood populated with low battery resources. The authors of [2] suggested a system in which all packet originators attach a payment token to each packet they send. Each intermediary on the packet's path to a base station then verifies whether this token is a *winning ticket* for him. This outline is based on the probabilistic payments suggested by Rivest [4]. Intermediaries with winning tickets can send a *reward claim* to their accounting center in order

to be rewarded for their work. The cost paid by the originator covers – on average – the cost of routing and other network maintenance. Therefore, base stations receive two kinds of packet: reward claims that they send to the accounting centers and packets with payment tokens. In the latter, the base stations send the packets (without the token) to the expected destination and the tokens are sent to the accounting centers. Packets with invalid tokens are dropped, as the transmission cannot be charged to anybody. The packet transmission procedure is detailed in Section 2.4 and the reward protocol is described in Section 2.5.

2.3 Setup

When a user registers for access to the home network, he is assigned an identity u and a symmetric key K_u. The pair (u, K_u) is stored by both the user and the user's home network. From a routing point of view, each user u manages a list $\lambda_u = ((u_i, d_i, L_i))_i$ where u_i is the identity of a neighbor, d_i its path length (in terms of hops) to the closest base station and L_i its threshold for forwarding packets as explained later. λ_u is increasingly sorted according to d_i and then L_i.

2.4 Packet transmission protocol

Origination. The originator u_o of the packet p performs the following procedure.

1. Selects the reward level $L \in [0, \max_L]$.
2. Computes $\mu = \mathrm{MAC}_{K_{u_o}}(p, L)$ where MAC is a keyed-hash function.
3. Sends the tuple $P = (L, p, u_o, \mu)$ according to the Transmission procedure.

Transmission. In order to send a tuple $P = (L, p, u_o, \mu)$, a user u (originator or intermediary) performs the following procedure.

1. If the base station can be reached in a single hop then u sends P directly to it. If not, he goes to Step 2.
2. u selects the first entry (u_i, d_i, L_i) from λ_u for which $L_i \leq L$. If such an entry does not exist then u drops the packet.
3. u sends a *forward request* to u_i containing the reward level L.
4. If u receives an acknowledgment from u_i before a timeout δ, then he sends P to u_i. If not, he goes back to Step 2 to the next entry in λ_u.
5. If u is not the originator of the packet, he carries out the Reward protocol.

Acceptance by an intermediary. When a user u' receives a forward request from a user u with a reward level L, he agrees to forward the packet if and only if $L_{u'} \leq L$. If this is the case, he sends an acknowledgment to u and waits for the packet. He then carries out the Transmission procedure.

Acceptance by a base station.

1. When a packet $P = (L, p, u_o, \mu)$ is received by a base station in the originator's home network, the base station checks whether $\mu = \mathrm{MAC}_{K_{u_o}}(p, L)$ with the stored secret key K_{u_o}. If the check fails the packet is dropped; if not, μ is sent to the accounting center and p is sent to the closest base station to the recipient user. This base station broadcasts the packet to the recipient user.
2. When a packet $P = (L, p, u_o, \mu)$ is received by a foreign base station, the latter forwards it to the registration center of the originator's home network. This center performs the tasks described in the first step of this procedure.

2.5 Reward protocol

Recording. After a user u has forwarded a tuple $P = (L, p, u_o, \mu)$, he verifies whether $f(\mu, K_u) = 1$ where f is a given function described in Section 2.6. If the check succeeds, we can say that the user has a *winning ticket* and can claim a reward for this ticket. In this case, he records (u_1, u_2, μ, L) where u_1 is the identity of the user from whom he received the packet and u_2 is the identity of the user (or base station) to whom he sent the packet. Let M be the list of recorded reward 4-tuples.

Sending. When the user is able to reach the base station with only one hop, he sends the claim (u, M, m) directly to it, where $m = \mathrm{MAC}_{K_u}(\mathrm{hash}(M))$. If not, the claim is sent to the base station using the same procedure as a usual packet. Note that the list M is encrypted with the key of the user in both cases.

An example of packet transmission and reward claims is given on Fig. 1.

2.6 Winning function

The winning function f determines whether a ticket μ is winning for a user u. Let K_u be the secret key of u. μ is a winning ticket for u if and only if $f(\mu, K_u) = 1$. Since the attack described in Section 4 exploits this function, its design should be defined with care.
Jakobsson *et al.* suggest that this function could be a one-way hash function, but they say that such a function is too costly. Instead, they suggest choosing f such that $f(\mu, K_u) = 1$ if and only if the Hamming distance between μ and K_u is less than or equal to a threshold h, because this function is very lightweight. The authors of [2] note that if the list of recorded reward 4-tuples M is not encrypted, then an attack could be possible. We describe such an attack in Section 4.1 that results in the discovery of all secret keys if M is not encrypted, with only η requests to an oracle, where $152 \leq \eta \leq 339$ in practice. We then show in Section 4.3 that such an attack remains possible even when M is encrypted. In this case, the complexity of the attack depends on the implementation and the victim's environment, but it remains proportional to η.

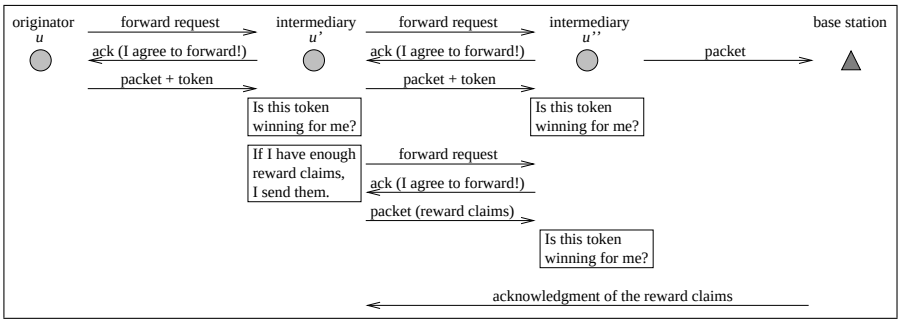


Fig. 1. Example of packet forwarding

u sends a packet. u' and then u'' agree to forward it. The token is winning for u' and he has enough reward claims to send them to the base station. u'' agrees to forward the reward claims. The base station acknowledges the reception of the reward claims.

2.7 Accounting and auditing

The scheme described in [2] relies on an accounting and auditing center. We assume for the sake of simplicity that these two entities are one and the same, along with the registration center. We call it the *operator center*. Note that there is only one operator center per network. The accounting center receives both user claims and transmission transcripts, both forwarded by the base stations. The accounting center periodically verifies all received user claims concerning all the recorded reward tuples it has received from base stations. All recorded originators are charged a usage fee according to their service contract. Moreover, the accounting center credits all parties (except the originator and the base station) whose identity appears in the accepted reward claim. Here, a reward claim is said to have been accepted if it is *correct*, i.e., if $f(\mu, K_u) = 1$ and a base station has reported the packet associated with the ticket μ as having been transmitted. The goal of the auditing center is to detect attacks in the network using statistical methods. According to Jakobsson *et al.*, the following attacks can be detected using the auditing techniques (except the *tampering with claims* attack which is prevented by the used of authentication methods): *Selective Acceptance*: The user agrees to receive (with the intent to forward it) a packet if and only if it contains a *winning* ticket; *Packet Dropping*: The user agrees to receive packets but does not forward them – whether he claims credit for winning tickets or not; *Ticket Sniffing*: A user claims credit for a packet he intercepted, but neither agrees to forward nor actually forward it. A more serious attack consists of users along a fake path submitting claims as if they had routed the packet; *Crediting a Friend*: A user with a winning ticket claims to have received the packet from (or have send it to) a user other than the true one; *Greedy Ticket Collection*: A user claims credits in excess of those specified by the protocol, by collecting and

sharing tickets with colluders; *Tampering with Claims*: A user modifies or drops the reward claim filed by somebody else in order to increase his profits or to remove harmful auditing information; *Reward level Tampering*: A packet carries an exaggerated reward level along its path, but the reward level is reduced before it is transmitted to the base station; *Circular Routing*: The packet transits through a circular routing in order to increase the benefit to the intermediaries; *Unnecessary Long Path Routing*: The packet transits through an unnecessary long path within a particular neighborhood in order to increase the benefit to the intermediaries since they have a valid ticket.

Our goal in this paper is not to discuss this technique. We assume that in the outcome, this statistical method fulfills the claims of the authors.

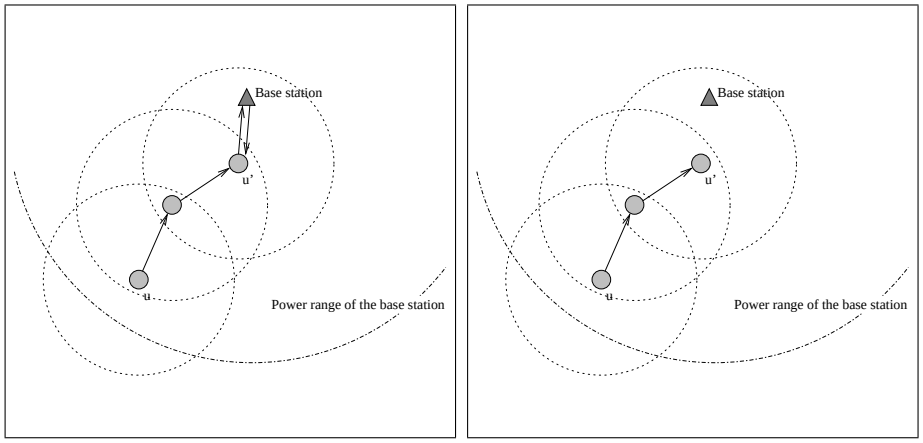
3 Communicating freely in a cell

In this section, we describe an attack which allows two misbehaving users in the same cell to communicate freely. We will show that this attack can be put into practice rather easily. We will then suggest a lightweight solution to counteract the threat.

3.1 Description of the attack

This attack consists of two users in the same cell communicating freely using fake identities, thus their neighbors will not be rewarded for their work. Firstly we recap that if a user u sends a message to a user u' who is not in his neighborhood, then the packet is sent to the base station through other users. Note that if u' is on the path from u to the base station, then he should not keep the packet when he receives it, but should forward it to the base station and wait for the packet to come back from the base station (see Fig. 2a). Unfortunately, there is no mechanism to protect against adversary wanting to take the packet on the uplink, as represented on Fig. 2b. Such fraudsters would not be punished since they are not registered to the accounting center; the weak point being that there is no authentication between the users on the packet path.
Note that it is rather easy for u' to be on the packet path, by claiming a fake distance from the base station and a fake reward level. In particular, if two hops are enough to link u and u', the attack will definitely succeed: u' announces in his neighborhood that he is able to reach the base station with only one hop even if it is untrue. Due to this unfair competition, his neighbors will choose him to route the packets[1]. Better still, the recipient attacker can be "near" the path (i.e., she can eavesdrop the transmitted data without being on the routing path) or "near" the base station and she can then hijack the packet without even being on the packet routing path. Even if this latter case is not scalable, it is realistic because the attack is easy to put into practice. For instance, if one of the attackers lives

[1] This misbehavior could also be used to set up a "famine" attack against a node.



(a) Well-behavior: u' forwards the packet to the base station and waits for it to come back.

(b) Misbehavior: u' keeps the packet when he receives it instead of forwarding it to the base station.

Fig. 2. u sends a packet to u'

close to a base station, she can communicate freely in her cell, hijacking the packets which are intended to her. Punishing her is not straightforward because her identity does not appear in the packet and she participates only passively in the attack. Note, however, that the attack is possible on the uplink, but not on the downlink.

3.2 Fixing the scheme

Fixing the scheme without requiring heavy cryptographic functions – which the authors sought to avoid – is a difficult task because the attack relies on the fact that there is no authentication between the users. One way to fix the scheme is to oblige the packet to pass through the base station in order to be usable by the recipient. This can be done if each node on the uplink encrypts the packets that it forwards – with a key also known by the base station – using symmetric encryption which is much less expensive than asymmetric encryption. Thus, each node can be sure that the packet will have to be decrypted by the base station otherwise it will be rendered unusable for the recipient.
However, such a solution is quite costly. We suggest instead relaxing the security requirements. Indeed, since [2] is based on the fact that while a small amount of fraud is acceptable, large-scale fraud has to be avoided, we suggest reducing the number of computations by introducing a probabilistic mechanism: each user encrypts the packet with a probability ρ. If n is the number of intermediaries between the two attackers, ρ is the probability that an intermediary encrypts

the packet, and τ is the probability that the attack succeeds, then we have: $\tau = (1-\rho)^n$. Taking, for example $n = 5$, which seems a realistic value and $\rho = \frac{1}{2}$, we have $\tau \approx \frac{3}{100}$. We may even determine a threshold at which the attack is no longer an attractive proposition[2] and consequently decrease ρ until it reaches this threshold. This technique substantially reduces the computations performed by the nodes. If a node decides to reduce ρ in order to save its battery power, it will be detected by the auditing center, since its rate of forwarded encrypted packets over the total number of forwarded packets will be abnormally low.

4 Recovering secret keys using side channel attack

As discussed in Section 2, the goal of the secret keys stored by the nodes is twofold. Firstly, these keys aim to encrypt the reward claims. Secondly, they are used to charge the originator of packets: the originator's secret key is used to compute a MAC on the packet, which is used by the accounting center in order to charge the owner of the key. In other words, if an attacker is able to steal a secret key, he is able to communicate freely and the charged node is the owner of the stolen key. We will show in this section how an attacker can carry out such an attack. For the sake of simplicity, we will first give a theoretical overview of the attack, showing that if an attacker can access an oracle, defined below, then he can recover the 128 bit keys using only approximately a few hundred oracle requests. We will then show in Section 4.3 that such an oracle is available in practice. Finally we fix the scheme using a keyed-hash function.

4.1 Description of the attack: theoretical approach

Firstly, we will recap the principle of the winning tickets. A user sends a tuple $P = (L, p, u_o, \mu)$ to a user u where $\mu = \mathrm{MAC}_{K_{u_o}}(p, L)$; L, p, u_o, and K_{u_o} have already been defined in Section 2.4. u checks whether $f(\mu, K_u) = 1$ that is $d_{\mathcal{H}}(\mu, K_u) \leq h$ where $d_{\mathcal{H}}$ represents the Hamming distance, h is a given threshold, and K_u is the secret key of u.
We assume in this theoretical approach that if the test succeeds then u sends the claim (u_1, u_2, μ, L) to the accounting center[3]; if the test fails, u sends nothing (see Fig. 3). Obviously, the intermediary nodes do not know K_{u_o}, therefore they are not able to check whether $\mathrm{MAC}_{K_{u_o}}(p, L)$ is valid. Thus, a node can be seen as an Oracle $\mathcal{O}$, such that for a request $\mu \in \{0,1\}^\ell$ where ℓ is the size of the secret key, $\mathcal{O}$ returns *true* if $d_{\mathcal{H}}(\mu, K_u) \leq h$ otherwise we consider that it returns *false*. We will now show that some information on the secret key leaks from the oracle. In other words, by sending some forward requests to a node and by spying on

[2] The cheater can repeat his attack until it succeeds but if ρ is small, the attack will no longer be attractive due to excessive battery consumption and the delay caused by repeated attempts.

[3] In practice, a claim is not sent as soon as a winning ticket is received, but they are recorded and then encrypted to be sent to the accounting center. We will consider the practical aspects in Section 4.3.

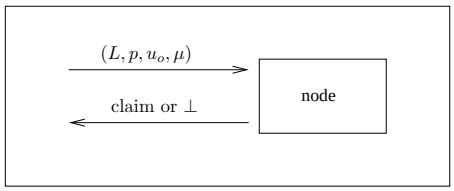


Fig. 3. The node can be seen as an oracle

its reward claims, an attacker can determine its secret key. The attack consists of two steps:

1. the first step aims to find a value $\hat{\mu} \in \{0,1\}^\ell$ such that $d_{\mathcal{H}}(\hat{\mu}, K_u) = h$ or $h+1$;
2. the second step aims to recover K_u by sending requests to $\mathcal{O}$ with slight variations of $\hat{\mu}$.

Let us denote $\mathcal{O}(\mu)$ the Boolean answer from the oracle for the request μ; so $\mathcal{O}(\mu)$ means that the answer is *true* and $\neg\mathcal{O}(\mu)$ means that the answer is *false*. Let $\mathcal{F}_{\text{in}} := \{\mu \in \{0,1\}^\ell \mid d_{\mathcal{H}}(\mu, K) = h\}$, $\mathcal{C}_{\text{in}} := \{\mu \in \{0,1\}^\ell \mid d_{\mathcal{H}}(\mu, K) \leq h\}$, $\mathcal{F}_{\text{out}} := \{\mu \in \{0,1\}^\ell \mid d_{\mathcal{H}}(\mu, K) = h+1\}$, and $\mathcal{C}_{\text{out}} := \{\mu \in \{0,1\}^\ell \mid d_{\mathcal{H}}(\mu, K) \geq h+1\}$. Let μ_i be the i-th bit of μ and $\mu^{(i)}$ be equal to μ except μ_i which is flipped. We assume in the sequel that $0 < h < \ell$.

Step 1 In order to solve the first step of the attack, we supply a Las Vegas algorithm (see Alg. 1), with parameters s, t, and μ, which allows to find a value μ on the border $\mathcal{F}_{\text{in}}$ or $\mathcal{F}_{\text{out}}$. Its principle is the following: given a random value μ, it puts a request to the oracle with this value and then it chooses (possibly randomly) s bits of μ if $\mathcal{O}(\mu)$ (resp. t bits of μ if $\neg\mathcal{O}(\mu)$), $r_1, r_2, \ldots, r_{s \text{ or } t}$, and sends $\mu^{(r_1)}, \mu^{(r_2)}, \ldots, \mu^{(r_{s \text{ or } t})}$ to the oracle. We assume that the parameters s and t are such that $(s,t) \neq (0,0)$. Let $\xi(\ell, h, s, t)$ be the probability that Alg. 1 answers. We have:

$$\xi(\ell, h, s, t) := A(\ell, h, s) \Pr(\mu \in \mathcal{C}_{\text{in}}) + B(\ell, h, t) \Pr(\mu \in \mathcal{C}_{\text{out}}) \tag{1}$$

where

$$\begin{aligned}
A(\ell, h, s) &= \Pr(\text{Alg. 1 answers} \mid \mu \in \mathcal{C}_{\text{in}}) \\
&= \Pr(\text{Alg. 1 answers} \mid \mu \in \mathcal{F}_{\text{in}}) \\
&= 1 - \binom{h}{s} / \binom{\ell}{s} \text{ if } s \leq h \text{ and 1 otherwise.} \\
B(\ell, h, t) &= \Pr(\text{Alg. 1 answers} \mid \mu \in \mathcal{C}_{\text{out}}) \\
&= \Pr(\text{Alg. 1 answers} \mid \mu \in \mathcal{F}_{\text{out}}) \\
&= 1 - \binom{\ell - h}{t} / \binom{\ell}{t} \text{ if } t \leq \ell - h \text{ and 1 otherwise.}
\end{aligned}$$

Lemma 1. *Given a random $\mu \in \{0,1\}^\ell$, the probability that $\mu \in \mathcal{F}_{in}$ is $\frac{1}{2^\ell}\binom{\ell}{h}$ and the probability that $\mu \in \mathcal{F}_{out}$ is $\frac{1}{2^\ell}\binom{\ell}{h+1}$.*

Proof. The proof is straightforward since $\mid \mathcal{F}_{\text{in}} \mid = \binom{\ell}{h}$, $\mid \mathcal{F}_{\text{out}} \mid = \binom{\ell}{h+1}$, and $\mid \{0,1\}^\ell \mid = 2^\ell$. □

From Lemma 1, we deduce that the probability that Alg. 1 answers is

$$\xi(\ell,h,s,t) = \frac{\binom{\ell}{h}}{2^\ell} A(\ell,h,s) + \frac{\binom{\ell}{h+1}}{2^\ell} B(\ell,h,t) \tag{2}$$

From the Las Vegas algorithm, it is straightforward to design a Monte Carlo algorithm as represented on Alg. 2. Let $C(\ell,h,s,t)$ be the number of rounds of Alg. 2 in order to find a value on the border; we have:

$$\Pr(C(\ell,h,s,t) = c) = \xi(1-\xi)^{c-1} \text{ if } c > 0 \text{ and } \Pr(C(\ell,h,s,t) \leq 0) = 0.$$

We compute the *average number of rounds* of Alg. 2, $\tilde{C}(\ell,h,s,t)$, in order to

Alg. 1: Find-Border-Las-Vegas(s,t,μ)

send μ to the oracle $\mathcal{O}$
if $\mathcal{O}(\mu)$ **then** $b \leftarrow s$
else $b \leftarrow t$
end
pick b distinct random r_i in $[1,\ell]$
send $\mu, \mu^{(r_1)}, \mu^{(r_2)}, \ldots, \mu^{(r_b)}$ to $\mathcal{O}$
if $\mathcal{O}(\mu) \wedge \neg \bigwedge_{i=1}^{i=b} \mathcal{O}(\mu^{(r_i)})$
  then return "μ is in $\mathcal{F}_{\text{in}}$"
else
  if $\neg\mathcal{O}(\mu) \wedge \bigwedge_{i=1}^{i=b} \mathcal{O}(\mu^{(r_i)})$
    then return "μ is in $\mathcal{F}_{\text{out}}$"
  else return $\perp$
  end
end

Alg. 2: Find-Border-Monte-Carlo(s,t)

pick a random value $\mu \in \{0,1\}^\ell$
if Find-Border-Las-Vegas$(s,t,\mu) \neq \perp$
  then return μ
else
  iterate Find-Border-Monte-Carlo(s,t)
end

complete the first step of the attack.

$$\tilde{C}(\ell,h,s,t) = \lim_{k\to\infty} \sum_{c=1}^{k} c\,\xi(1-\xi)^{c-1} = \frac{\xi}{(1-(1-\xi))^2} = \frac{1}{\xi}. \tag{3}$$

Given that each round of Alg. 2 requires either $t+1$ (with probability σ) or $s+1$ (with probability $1-\sigma$) calls to the oracle, and that $(s,t) \neq (0,0)$, we compute from (2) and (3) the *average number of requests to the oracle* in order to complete the first step of the attack:

$$\frac{2^\ell(1+s\sigma+t(1-\sigma))}{\binom{\ell}{h}A(\ell,h,s) + \binom{\ell}{h+1}B(\ell,h,t)}. \tag{4}$$

Step 2 We now consider the second step of the attack, whose complexity is *a priori* ℓ according to Lemma 2.

Lemma 2. *Given $\mu \in \mathcal{F}_{in}$ (or given $\mu \in \mathcal{F}_{out}$), we can recover the key K with only ℓ requests to the oracle $\mathcal{O}$.*

Proof. We assume that we have $\mu \in \mathcal{F}_{\text{in}}$; we know therefore that $\mathcal{O}(\mu)$ is true. For every i ($1 \leq i \leq \ell$) we ask the oracle the question $\mathcal{O}(\mu^{(i)})$. We have $\mu_i = K_i$ if and only if $\mathcal{O}(\mu^{(i)})$ is false, that is flipping μ_i moves away μ from the key K. The same track is used to prove the lemma with $\mu \in \mathcal{F}_{\text{out}}$.

□

Practically, the number of requests to the oracle in Step 2 can be reduced by (a) exploiting the values already checked in the first step of the attack, (b) using the fact that the second step can be halted as soon as the h or $h+1$ bits that differ from the key have been found (the other bits can thus be found by inference). We describe these two points in further detail below.

(a) We can re-use the requests of the last round of Alg. 1. in the second step of the attack. We thus have s answers (resp. t answers) from the oracle if $\mu \in \mathcal{C}_{\text{in}}$ (resp. $\mu \in \mathcal{C}_{\text{out}}$). $s\sigma + t(1-\sigma)$ answers from the oracle are thus already known on average.

(b) Since the second step flips the bits of μ independently, one after the other, the process can be halted as it has found the h (resp. $h+1$) bits μ_i s.t. $\mu_i \neq K_i$ or the $\ell - h$ bits s.t. $\mu_i = K_i$ (resp. $\ell - h - 1$) when $\mathcal{O}(\mu)$ (resp. $\neg\mathcal{O}(\mu)$). We denote $\zeta(\ell, h)$ the average number of calls to the oracle that can be saved using this inference method. We compute $\zeta(\ell, h)$ below. We notice that the process stops at the round i if and only if

$$\mathcal{O}(\mu^{(i)}) \neq \mathcal{O}(\mu^{(i+1)}) = \cdots = \mathcal{O}(\mu^{(\ell)}).$$

We have to consider the case where $\mu \in \mathcal{F}_{\text{in}}$ and the case where $\mu \in \mathcal{F}_{\text{out}}$. Let us begin with $\mu \in \mathcal{F}_{\text{in}}$. Let $Y_{\text{in}} := \ell - i$. We have

$$\begin{aligned}
\Pr(Y_{\text{in}} = 1) &= \frac{\ell - h}{\ell - 1} \cdot \frac{h}{\ell} + \frac{h}{\ell - 1} \cdot \frac{\ell - h}{\ell} \\
\vdots \qquad & \qquad \vdots \qquad \qquad \vdots \\
\Pr(Y_{\text{in}} = i) &= \frac{\ell - h}{\ell - i} \cdot \prod_{\substack{j=0 \\ j<\ell}}^{i-1} \frac{h - j}{\ell - j} + \frac{h}{\ell - i} \cdot \prod_{\substack{j=0 \\ j<\ell}}^{i-1} \frac{\ell - h - j}{\ell - j}
\end{aligned}$$

So, the average number of requests that can be save if $\mu \in \mathcal{F}_{\text{in}}$ is:

$$\tilde{Y}_{\text{in}}(\ell, h) = \sum_{\substack{i=1 \\ i<\ell}}^{\infty} i \cdot \left(\frac{\ell - h}{\ell - i} \cdot \prod_{j=0}^{i-1} \frac{h - j}{\ell - j} + \frac{h}{\ell - i} \cdot \prod_{j=0}^{i-1} \frac{\ell - h - j}{\ell - j} \right).$$

If $h \approx \frac{\ell}{2}$, we can estimate $Y_{\text{in}}(\ell, h)$ using a geometric probability law with parameter $\frac{1}{2}$, from which we obtain

$$\tilde{Y}_{\text{in}}(\ell, h) \approx 2. \tag{5}$$

We define Y_{out} using the same method, and prove that $\tilde{Y}_{\text{out}}(\ell, h) \approx \tilde{Y}_{\text{in}}(\ell, h)$. Thus, the average complexity of the second step is $\ell - s\sigma - t(1-\sigma) - \zeta(\ell, h)$, what can be approximated by

$$\ell - s\sigma - t(1-\sigma) - 2. \tag{6}$$

From (4) and (6) we obtain when $(s,t) \neq (0,0)$ the complexity of the attack[4] in terms of requests to the oracle:

$$\frac{2^\ell(1 + s\sigma + t(1-\sigma))}{\binom{\ell}{h}A(\ell,h,s) + \binom{\ell}{h+1}B(\ell,h,t)} + \ell - s\sigma - t(1-\sigma) - 2. \tag{7}$$

4.2 Interpretation

If ℓ denotes the size of K_u in the usual binary representation, then the probability that μ is a winning ticket is $\sigma = \frac{1}{2^\ell}\sum_{i=0}^{h}\binom{\ell}{i}$. In order to envisage an intuitive representation of the probability that a ticket is winning, we draw on Fig. 4 the probability according to the threshold h when $\ell = 128$. In practice, the probability of a ticket being winning should be greater than $\frac{1}{100}$ otherwise the nodes would not collaborate: this implies that h should be greater than 51. Fig. 5

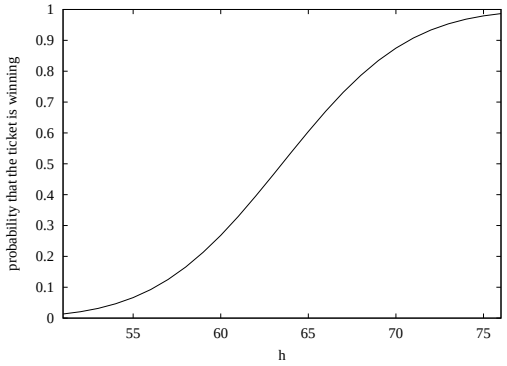


Fig. 4. Probability that a randomly chosen ticket is winning according to the value h

represents the corresponding complexity of the attack, given by (7), when s and t are optimal. In the range of the practical values of h, the attack requires between 152 and 339 requests to the oracle in order to recover a key 128 bits long! Optimal values for s and t depend on h. When $h = 51$, the optimal complexity is obtained where $s = 5$ and $t = 0$.

[4] Note that the algorithm which is given here aims to demonstrate that a practical attack is possible (if we can use oracle $\mathcal{O}$) but more sophisticated algorithms could further reduce the complexity. For instance, the attack can be improved if the calls to Alg. 1 are not independent. Indeed, consider the case where Alg. 1 is used with $t = 1$ and the two requests to the oracle are μ and $\mu^{(i)}$. If the protocol does not

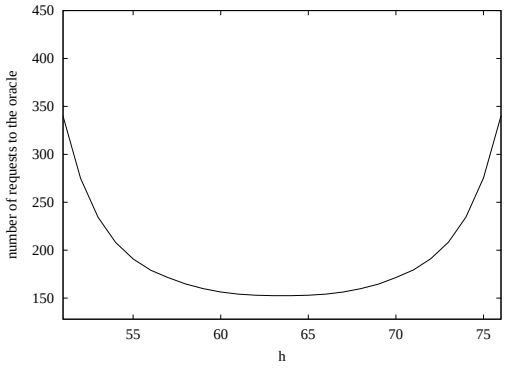


Fig. 5. Number of requests to $\mathcal{O}$ in order to recover the secret key of a given user

4.3 The attack in practice

In [2], the nodes do not send a reward claim as soon as they receive a winning ticket. Instead, they store them in a list M which is encrypted before being sent to the accounting center. Consequently, an attacker is no longer able to match the values submitted to the node (requests to the oracle) with the sent claims (answers from the oracle). In other words, she no longer knows which of her value will generate a claim. Unfortunately, the attacker can still use an oracle even in this case; the attack consists of disturbing the input distribution of the node by sending beams of equal random values μ and then by analyzing whether the output distribution is disturbed, i.e., if the list of the reward claims is longer[5] or sent more frequently than usual. Indeed, if the random value μ forming the beam is such that $f(\mu, K) \neq 1$, then the output will not be disturbed. If not, the number of claims will be larger (length of packets larger than usual or packets more frequent than usual), meaning that the oracle answers true for this value. The length of the beam depends on the node environment and on the implementation of the protocol. Note that it is not necessary for the beam to fill the buffer; it merely has to sufficiently disturb the input distribution in such a way that the disturbance is detectable in the output distribution. Thus, the remainder of the buffer can be filled with random values. In this way, other (honest) users requesting the node to forward are helping the attacker by filling the buffer! Consequently, depending of the environment and implementation, the complexity of the attack remains proportional to the theoretical complexity given in Section 4.1.

---

answer, we clearly have $\mathcal{O}(\mu) = \mathcal{O}(\mu^{(i)})$. We know, however, that if $\mathcal{O}(\mu)$ then $\neg\mathcal{O}(\overline{\mu})$ where $\overline{\mu}$ means that all the bits of μ have been flipped. We can now use this new value for the next call to Alg. 1, thus decreasing the number of calls to the oracle.

[5] [2] says that, even if an attacker cannot distinguish which ticket generates a reward claim using their approach, she can determine how many reward claims are sent.

4.4 Fixing the scheme

Since the node has a buffer to store the winning tickets, one may think that in order to fix the scheme, it could reject random values that have already been stored. Indeed, the attacker is no longer able to send beams of equal random value. Unfortunately, it is possible to perform our attack in another way as follows. We assume w.l.o.g that the buffer is empty at the beginning of the attack[6]; for the sake of simplicity, we also assume that there are no other users making requests to the victim during the attack[7]. The attacker sends the victim the following beam:

$$\alpha^1, \alpha^2, \ldots, \alpha^{n-1}, \alpha^n$$

where the α^i are independent random values, until the node sends its list of claims (i.e., the oracle responds). In this case, the last sent value α^n is such that $f(\alpha^n, K) = 1$. The attacker wants now to check the value μ and sends for that the beam:

$$\alpha^1, \alpha^2, \ldots, \alpha^{n-1}, \mu$$

The $n-1$ first values fill the buffer, except the last space. Consequently either $f(\mu, K) = 1$ and therefore the node will send its claims or else $f(\mu, K) \neq 1$ which implies no answer from the node.

We feel that, whatever the patches applied, computing the Hamming distance between the secret key and another value in order to determine the winning tickets is not a good idea. The way that we suggest to fix the scheme consists of modifying the protocol such that the information that an attacker can obtain with the attack is rendered useless. Thus, we propose that a ticket is winning for u if and only if:

$$d_{\mathcal{H}}(\mu, \mathrm{hash}(K_u)) \leq h.$$

This technique has two advantages: when K_u is kept in a tamperproof memory, only $\mathrm{hash}(K_u)$ remains in the vulnerable memory; the attacker is able to obtain $\mathrm{hash}(K_u)$, but the only thing that the attacker can do with this information is a *greedy ticket collection attack*, which is detected by the auditing center (see Section 2.7). Note that in [2], even if the key is encrypted with a password when the node is turned off, it has to remain permanently in clear in the non-tamperproof memory when the node is turned on. The second advantage is the lightweight of this solution because the hash value is computed only once instead of being computed for every packet.
If the computational capabilities of the nodes allow a keyed-hash function to be

[6] The list is empty as soon as the user sends his list of claims. Note that even if the size of the buffer is not fixed, an attack is possible.

[7] This is not actually a problem since the attacker can accept the request instead of the victim, as explained in Section. 3.1, or if it is not possible, this disturbance will slightly increase the complexity of the attack, but the attack will still be possible – remember that the probability that the value of a "disturbing" requester generates a winning ticket is very low.

carried out for each packet, then a more secure way would be to decide that a ticket is winning if and only if:

$$d_{\mathcal{H}}(\mu, \mathrm{MAC}_{\mathrm{hash}(K_u)}(\mu)) \leq h.$$

Note that again it is not the key itself which is in the vulnerable memory, but only the hash of the key. If one of these solutions is used to fix the protocol, the attacker can no longer use the node as an oracle.

5 Conclusion

In this paper, we have analyzed the security of the micro-payment scheme designed for asymmetric multi-hop cellular networks proposed by Jakobsson, Hubaux, and Buttyán. We have shown that the security of the scheme is compromised. Our contribution has mainly consisted of showing two attacks that entirely break the system in the sense that all the users' secret keys can be determined, with only a few hundred trials. This implies that an attacker can thus communicate freely, without being charged: the owners of the stolen keys are charged instead. We have suggested some lightweight but efficient modifications in order to repair the micro-payment scheme.

Acknowledgments

The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. I would like to thank M. Jakobsson, J.-P. Hubaux, and L. Buttyán, for their relevant comments on this work, as well as N. Ben Salem and S. Vaudenay for their helpful discussions.

References

1. Thomas Haug. Overview of GSM: philosophy and results. *International Journal of Wireless Information Networks*, 1(1):7–16, 1994.
2. Markus Jakobsson, Jean-Pierre Hubaux, and Levente Buttyán. A micro-payment scheme encouraging collaboration in multi-hop cellular networks. In *Financial Cryptography – FC'03*, vol. 2742 of *LNCS*, pp. 15–33, Le Gosier, Guadeloupe, French West Indies, January 2003. IFCA, Springer.
3. Ying-Dar Lin and Yu-Ching Hsu. Multihop cellular: A new architecture for wireless communications. In *Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies – INFOCOM 2000*, vol. 3, pp. 1273–1282, Tel-Aviv, Israel, March 2000. IEEE.
4. Ronald Rivest. Electronic lottery tickets as micropayments. *Financial Cryptography – FC'97*, vol. 1318 of *LNCS*, pp. 307–314, Anguilla, British West Indies, February 1997. IFCA, Springer.