Improved Anonymity for Key-trees*

Michael Beye[1] and Thijs Veugen[1,2]

[1] Information Security and Privacy Lab, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, The Netherlands
M.R.T.Beye@tudelft.nl
[2] Security Group, TNO, The Netherlands
thijs.veugen@tno.nl

Abstract. Randomized hash-lock protocols for Radio Frequency IDentification (RFID) tags offer forward untraceability, but incur heavy search on the server. Key trees have been proposed as a way to reduce search times, but because partial keys in such trees are shared, key compromise affects several tags. Buttyán et al. have quantified the resulting loss of anonymity in the system, and proposed specific tree properties in an attempt to minimize this loss. We will further improve upon these results, and provide a proof of optimality. Finally, our proposals are compared to existing results by means of simulation.

Key words: RFID, Hash-lock protocol, key-tree, anonymity, anonymity set

1 Introduction

We consider the problem of authenticating many Radio Frequency IDentification (RFID) tags through randomized hash-lock protocols, in an efficient way. The tags are authenticated towards the reader through a challenge-response mechanism. Each tag authenticates itself using some secret key combined with a random value (*nonce*). To authenticate the tag, the reader will have to check the keys of all tags combined with all possible random values, in order to find a match. Since this task is very intensive for the reader, a key-tree is used. Each leaf of the tree represents a tag, and each edge corresponds to a specific key. Every tag is assigned the keys that lie on its path from the root of the tree (see Fig. 1). During the authentication protocol, a tag is authenticated step by step, i.e. edge by edge, such that the computational load of the reader, and thus the total authentication time, is lowered.

However, the authentication mechanism should still remain secure. If hardware-level tampering is taken into account, keys that were assigned to compromised tags can become known to the adversary. Because partial keys are shared between neighboring tags in the tree, several additional tags may be partially broken as

* Part of this research was performed at TNO for a master's thesis for the University of Utrecht (UU). Special thanks go to Gerard Tel (UU) for his advice, and to Harry Fluks (TNO) for his work on the simulation code.

well. How to construct the tree such that the number of (partially) broken tags will be minimal in case of one or more compromises?

This paper considers the trade-off between efficiency (minimizing authentication time), and security (minimizing the impact of tag compromise on neighboring tags), of such authentication mechanisms.

The layout of this paper is as follows: Section 2 will outline related work, with an emphasis on Buttyán et al.'s previous work on the optimization of key-trees. In Section 3, Buttyán's optimization problem is rephrased, an improved solution is suggested, and its effect is quantified. Finally, conclusions will be drawn in Section 4.

2 Related Work

Hash-chain protocols are meant to provide *forward untraceability*, by updating tag IDs in a one-way manner. This way, past IDs cannot be recovered, even through tampering. Examples are OSK (by Ohkubo, Suzuki and Kinoshita in [14]) and Yeo's protocol [16]. In [1], Avoine et al. suggest applying time-memory trade-offs (based on Hellmann tables [9]) to hash-chain protocols (namely OSK and an improved version thereof). In [2] they extend this trade-off with so-called *rainbow tables* and *checkpoints* to further improve efficiency. Hash-chain protocols have weaknesses, including *protocol exhaustion* (when the end of a chain is reached, tags can no longer update their IDs and become traceable) and *desynchronization* (server and tag chains can become out of synch if tags are queried by third parties).

A different class of hash-based authentication schemes called *Hash-lock protocols* (due to Weis et al.) was devised to solve the aforementioned problems. Tags are locked and unlocked, using hashes of their ID as the key. The *static hash-lock scheme* [15] is vulnerable to both *replay attacks* and *tracking*, but in the same paper, Weis et al. offer the *randomized hash-lock scheme* as a solution to such attacks: it adds *tag freshness* (a *nonce* generated by the tag) to prevent reader impersonation and tracking. The nonce is used as a challenge, and is hashed together with the tag's ID to form a one-time-use authentication key (the expected response). Juels and Weis [10] later added reader freshness to also prevent tag impersonation.

Note that precomputation cannot be used in these protocols, because the use of freshness makes the search space too large – one would need to compute values not only for each tag, but for each tag ID in combination with all possible nonces. Other solutions are required to reduce search complexity.

Molnar was the first to propose using a *tree of secrets* for RFID tags [11]. Although originally used for a system built around exclusive-OR and a pseudo-random function, it can be applied to other challenge-response building blocks. Damgård and Østergaard Pedersen [5] use the same concept, but speak of *correlated keys*. Nohara et al. in their "K-steps protocol" ([12], also dubbed NIBY) propose to apply trees to the hash-lock setting. They use the term *group IDs*

rather than correlated keys, and their trees are unconventional (being of non-uniform depth). Note that all these approaches use a sequence of group- and sub-group IDs to quickly and gradually narrow down a tag's identity. As Molnar mentions, *partial keys in such a tree should be chosen independently and uniformly from a key space of sufficient entropy.* Failure to do so would make the system vulnerable to attack. If partial keys are chosen properly, the adversary will have a large key space to search, while the owner of the system can efficiently search through a limited subspace (the actual tree).

The trade-off that exists between efficiency and security in tree-based protocols was already pointed out by Avoine [1], with respect to Molnar's original trees. Because tags share their partial keys, if one tag is compromised (i.e. has its memory probed through invasive tampering), an adversary learns partial keys for several other tags as well. This will enable him to decipher their responses in some of the verification steps, resulting in reduced anonymity and facilitating tracking. Nohl and Evans [13] try to quantify this more precisely. They distinguish between scenarios where compromised tags are chosen in a *selective* or a *random* way, and compute the *information leakage measured in bits.*

A paper of particular interest is by Buttyán et al. [4], where the concept of trees with *variable branching factors* is introduced, to better preserve anonymity in case of attack. Our work provides an optimization of Buttyán's solution.

In [8] Buttyán tries to further improve the balance between complexity and privacy in a new "group-based" authentication protocol. In short, the tags are divided into λ groups, where each group shares a group-key. Every tag also has an ID. This group-based scheme can be seen as a tree of depth 2, where every group-ID is tried, but the last stage (unique ID) only requires one decryption instead of exhaustive search. This means that the tree can be even wider at the top than a Buttyán tree, and thus attains a higher anonymity score. However, we choose not to follow this example because we believe that the group-based authentication protocol in [8] has inherent flaws, as will be explained in Section 4.

2.1 Notation

This paper bases its notation on that of Buttyán in [4], but makes minor extensions:

- $T = \{t_1, \cdots, t_N\}$: set of all tags in the system
- N: size of T, or actual number of tags in the system
- N': number of leaves in the tree ($\prod(B)$), or maximum number of tags in the system, $N' \geq N$
- c: number of compromised tags
- $P(t_i)$: helper function that returns the anonymity set to which tag t_i belongs
- P_j: anonymity set j, $0 \leq j \leq \ell$
- ℓ: number of anonymity sets in a given configuration
- S: size of a given anonymity set
- $\bar{S}$: average size over all anonymity sets in a given configuration

- $\bar{S}_{\langle - \rangle}(c)$: expected values of $\bar{S}$ (averaged over all configurations containing c compromised tags, see Definition 2)
- $\bar{S}_0(c)$: lower bound for $\bar{S}$, in the worst-case configuration containing c compromised tags (Definition 3)
- $B = (b_1, \ldots, b_d)$: a "branching factor vector" (or tuple), representing a tree; furthermore, $B \backslash \{b_1, \cdots, b_x\}$ denotes the vector $(b_{x+1}, \ldots, b_d)$
- d: depth of the tree
- $R(B)$: resistance to single member compromise for a tree with branching factor vector B. $R(B) \equiv \frac{\bar{S}_{\langle - \rangle}(1)}{N} \equiv \frac{\bar{S}_0(1)}{N}$
- $E(x)$: expected value of a variable x (weighted average of all possible values that this random variable can take on)
- $\sum(B)$: shorthand for $\sum_{i=1}^{d} b_i$, or the sum over all elements in B
- $\prod(B)$: shorthand for $\prod_{i=1}^{d} b_i$, or the product over all elements in B

2.2 Buttyán Trees

Buttyán et al. observed the time-anonymity tradeoff and noted that *narrow, deep trees allow faster search; it is wide, shallow trees that provide more anonymity.* Obviously, if many tags share the same partial keys, many tags can be excluded from the search space after each authentication stage, thus making search faster. The increased anonymity can be intuitively explained by the fact that when partial keys are shared between fewer tags, the amount of information gained by compromising a single tag is limited. Buttyán uses the concept of *anonymity sets* (Pfitzmann and Köhntopp [7], Díaz [6]) to quantify matters.

Definition 1. *Assume a tag t_i sends a given message m (or participates in a protocol execution). For an observer O, the anonymity set $P(t_i)$ contains all tags that O considers possible originators of m. Because all tags in $P(t_i)$ are indistinguishable to O, t_i is anonymous among the other tags in the set.*

Anonymity sets provide a sliding scale for anonymity, where belonging to a larger set implies a greater degree of anonymity. Total anonymity holds if the set encompasses all possible originators in the whole system (one is indistinguishable among all N tags in T), and belonging to a singleton set implies a complete lack of anonymity.

To measure the level of anonymity offered by a tree, Buttyán looks at the level of anonymity provided for a randomly selected member. This *expected size of the anonymity set that a randomly selected member will belong to*, is denoted $\bar{S}$. One could also view it as *the average anonymity set size over all tags*, as shown in (1). Note that $\bar{S}$ can be computed for any given scenario where a tree is broken into anonymity sets. Note that, for $c > 1$, the sizes of anonymity sets within the tree can vary, as different configurations of broken tags are formed. Configurations containing the same (number and size of) anonymity sets are considered identical, because sets can always be ordered in ascending order without loss of generality.

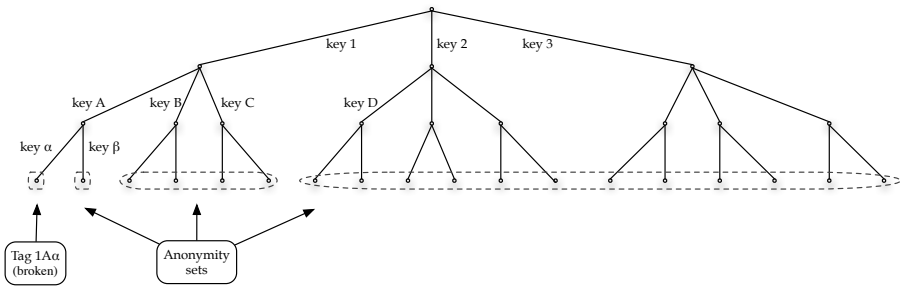

Fig. 1: Buttyán tree with a single broken tag

$$\bar{S} = \sum_{i=1}^{N} \frac{|P(t_i)|}{N} = \sum_{j=1}^{\ell} \frac{|P_j|}{N} |P_j| = \sum_{j=1}^{\ell} \frac{|P_j|^2}{N} \quad , \tag{1}$$

where $P(t_i)$ is a function that returns the anonymity set to which tag t_i belongs, P_j denotes an anonymity set and ℓ is the number of sets.

Buttyán then defines R, the *resistance to single member compromise*, as $\bar{S}$ computed for a scenario where *a single tag* is broken, and then normalizing the result (as in Díaz [6]). Note that because we can freely order the anonymity sets, $c = 1$ leads to a single unique configuration. With its range of $[0, 1]$, R is independent of N, allowing for easy comparison between systems of different sizes.

$$R = \frac{\bar{S}}{N} = \sum_{j=0}^{\ell} \frac{|P_j|^2}{N^2} = \sum_{i=0}^{d+1} \frac{|P_j|^2}{N^2} \quad , \tag{2}$$

where P_j denotes an anonymity set, ℓ is the number of sets, d denotes tree depth, and $\bar{S}$ *is computed for the (unique) scenario resulting from single member compromise*. Verify that, in this scenario, the number of sets ℓ is indeed equal to $d+1$.

Buttyán proposes the use of trees with different, independent branching factors on each level, sorted in descending order (as shown in Fig. 1). We will refer to such trees as "*Buttyán trees*", and to trees with a constant branching factor as "*Classic trees*".

Trees will be described by their branching factor vectors $B = (b_1, \ldots, b_d)$, where the variables b_i $(1 \leq i \leq d)$ are positive integers denoting the branching factor at level i.

Buttyán et al. in [4] reach the conclusion that the branching factors near the root contribute more to $\bar{S}$ and R. For trees with variable branching factors this means that a deep, top heavy Buttyán tree can potentially outperform a shallow classic tree.

Theorem 1 rephrases Buttyán's notion of an optimal $R(B)$ using the term "*lexicographically largest*"; the proof for the original version can be found in [4].

Theorem 1. *Let B and B' be two factor vectors, their elements sorted in descending order, s.t. $\prod(B) = \prod(B') = N$. If B is lexicographically larger than B', then $R(B) > R(B')$,*

Similarly, we rephrase Buttyán et al.'s optimization problem, which is centered around Theorem 1, as:

Problem 1. Given the total number N of members and the upper bound D_{max} on the maximum authentication delay, find the lexicographically largest vector $B = (b_1, \ldots, b_d)$ subject to the following constraints:

$$\prod(B) = \prod_{i=1}^{d} b_i = N, \text{ and } \sum(B) = \sum_{i=1}^{d} b_i \leq D_{max} \ . \tag{3}$$

Buttyán et al. provide a *greedy* algorithm that solves this problem recursively. It starts with the prime factorization of N and tries to combine prime factors as long as the sum (authentication time) remains acceptable.

However, Buttyán recognizes that trees need to stand up to more than single tag compromise. Without going into mathematical detail, Buttyán suggests to express $\bar{S}$ for the general case in two different ways:

Definition 2. *$\bar{S}_{\langle - \rangle}(c)$ expresses $\bar{S}$ as the average over all $\binom{N}{c}$ possible distributions of c compromised members across the tag set T.*

Our notation is a natural extension of Buttyán's $\bar{S}_{\langle - \rangle}$, directly incorporating c. Depending on how each successive member is picked from the tree, different anonymity sets are broken down. Buttyán notes that computing $\bar{S}_{\langle - \rangle}$ is hard, and therefore suggests an alternative measure:

Definition 3. *$\bar{S}_0(c)$ represents the* worst-case *value of $\bar{S}$ for all $\binom{N}{c}$ possible distributions of c compromised members across the tag set T.*

Although not stated explicitly in [4], this worst-case value is attained in (any of) the most uniform distributions of c compromised tags across T.

Proof. Assume that we are allowed to choose tags to be compromised sequentially, with the aim to minimize the average anonymity set size. The first compromised tag leads to a unique configuration. Each subsequent compromised tag leads to a new configuration, with more anonymity sets (of varying, decreasing size). To minimize the average set size in the *resulting* configuration, the next tag to be compromised should be chosen from (one of) the largest anonymity set(s) in the *current* configuration. When sorting anonymity sets in ascending order, we observe that this is equivalent to choosing tags (as) uniformly (as possible given the tree structure) across T. By induction, our claim holds for any c. □

Again, Buttyán's notation $\bar{S}_0$ is generalized to directly incorporate c. Buttyán correctly remarks that $\bar{S}_0(c)$ is far easier to compute, and acts both as a lower bound and an accurate approximation for $\bar{S}_{\langle - \rangle}(c)$.

3 Improved Key-trees

When considering the optimization problem as phrased by Buttyán (Problem 1), we first note that the condition $\prod(B) = N$ can lead to inferior solutions. Particularly when the number N has large prime factors, resulting in a small number of candidate branching factor vectors. We prefer the condition $\prod(B) \geq N$, which we will show leads to better results. An added advantage in practice is that it allows to maintain a small buffer of extra keys (see discussion in Section 3.1). Our optimization problem now becomes:

Problem 2. Given the total number N of members and the upper bound D_{max} on the maximum authentication delay, find the vector $B = (b_1, \ldots, b_d)$ that maximizes $R(B)$ subject to the following constraints:

$$\prod(B) = \prod_{i=1}^{d} b_i \geq N, \text{ and } \sum(B) = \sum_{i=1}^{d} b_i \leq D_{max} \ . \tag{4}$$

The anonymity measure $R(B)$ used here refers to the full tree with $\prod(B) = N'$ tags, of which exactly one is compromised, i.e. $c = 1$. Theorem 4 will later show that the same holds for the anonymity measure of the partial tree with $N \leq N'$ tags.

Theorem 2. *The maximal $R(B)$ under the constraints of Problem 2 is achieved by the lexicographically largest vector B that satisfies the constraints.*

The proof of Theorem 2 is given in the Appendix. The following theorem shows how to optimize the product of a branching vector, while keeping the sum constant and ignoring the lexicographic order. If $\sum(B) = b$, let us write the largest possible $\prod(B)$ as $\prod_b^{max}$.

Theorem 3. *Let $b \geq 2$ be a constant. Consider the set of branching vectors B (with elements in descending order) with $\sum(B) = b$. Then $\prod(B) = \prod_b^{max}$ holds when B is of one of the following forms: (3^*), $(4, 3^*)$ or $(3^*, 2)$, where 3^* denotes a sequence of branching factors 3 of arbitrary (possibly zero) length.*

Proof. Let B be a branching factor vector with $\sum(B) = b$. The proof is given by considering different cases.

Suppose B has a branching factor b_i equal to 1. Since $\sum(B) \geq 2$, there must be another branching factor b_j. Then, we could add b_i to b_j to increase $\prod(B)$ without modifying $\sum(B)$, meaning $\prod(B) \neq \prod_b^{max}$. Therefore, an optimal B (with $\prod_b^{max}$) contains no branching factor equal to 1.

Suppose B has a branching factor $b_i \geq 5$. Since $(b_i - 3) \cdot 3 > b_i$, we can increase $\prod(B)$ without modifying $\sum(B)$, by making an extra factor 3, meaning $\prod(B) \neq \prod_b^{max}$. Therefore, an optimal B contains only branching factors 2, 3 or 4.

Suppose B has two branching factors $b_i = b_j = 4$ $(i \neq j)$. Since $3 \cdot 3 \cdot 2 = 18 > 16 = 4 \cdot 4$, we can increase $\prod(B)$ without modifying $\sum(B)$ by changing

b_i and b_j to 3 and adding an extra 2, meaning $\prod(B) \neq \prod_b^{max}$. Therefore, the optimal B contains at most one branching factor 4.

Suppose B has two branching factors $b_i = b_j = 2$ $(i \neq j)$. Since $2 \cdot 2 = 4$, we could just as well substitute these branching factors by a single 4, making B lexicographically larger. Therefore, $\prod_b^{max}$ can be attained by at most one branching factor 2.

Suppose B has two branching factors $b_i = 2$ and $b_j = 4$. Since $2 \cdot 4 = 8 < 9 = 3 \cdot 3$, we can increase $\prod(B)$ without modifying $\sum B$ by substituting both factors by 3, meaning $\prod(B) \neq \prod_b^{max}$. Therefore, an optimal B will not contain both branching factors 2 and 4.

By considering these five cases, it follows that $\prod_b^{max}$ will be attained in one of the following cases:

1. B contains only 3's;
2. B contains one 4 and an arbitrary number of 3's;
3. B contains one 2 and an arbitrary number of 3's.

Consequently when $\sum(B) = b$, and we order the elements descendingly, $\prod_b^{max}$ will be attained by:

1. $B = (3^*)$, when $b \bmod 3 = 0$;
2. $B = (4, 3*)$, when $b \bmod 3 = 1$;
3. $B = (3*, 2)$, when $b \bmod 3 = 2$. □

When considering Problem 2, we know that when $b = D_{max}$ and $\prod_b^{max} < N$, there can be no solution that satisfies both constraints. On the other hand, when $\prod_b^{max} \geq N$, there is at least one solution. The obvious way to find the branching factors of the lexicographically largest solution, is to take a *greedy* approach. It means that the first branching factor is optimized first, then the second, etc. Algorithm 2 takes N and D_{max} as input and solves this problem recursively. Starting from $b_1 = D_{max}$, a branching factor b_1 is allowed, if a suitable tail (of one of three forms in Theorem 3) can be constructed with the remaining $D_{max} - b_1$, such that the product of B is large enough. If such a tail exists, it is optimized in recursion. If no suitable tail exists, b_1 is decremented. If no proper solution can be found at all (for $2 \leq b_1 \leq D_{max}$), an error is returned. This means that N is so large that even a binary tree does not allow search within the imposed D_{max}.

Note that because a tree is constructed only once, during a pre-computation stage, the runtime efficiency of the optimization algorithm is neither essential nor related to the authentication time. However, we note that the complexity of both Buttyán's algorithm and Algorithm 2 is linear (in N and/or D_{max}).

Since our search space is larger than Buttyán's, our optimal branching vector will either be equal to, or lexicographically larger than the output of Buttyán's algorithm, thus providing better anonymity. The potential difference in output can be illustrated with the help of the following examples:

– Buttyán's own example in Set 1 shows that Buttyán's algorithm is not optimal in the setting of Problem 2. The output of Algorithm 2 is lexicographically

```
int N, int D_max, vector B, vector B'
g(N, D_max)
{
    for (int b_1 = D_max; b_1 ≥ 2; b_1−−) do
    {
        h = ⌊(D_max − b_1)/3⌋
        if ((D_max − b_1) mod 3 == 0) then B = (b_1, 3^h)
        else if ((D_max − b_1) mod 3 == 1) then B = (b_1, 4, 3^h)
        else /**((D_max − b_1) mod 3 == 2)**/ B = (b_1, 3^h, 2)

        if (∏(B) ≥ N) then return B' = (b_1, g(N/b_1, D_max − b_1))
    }
    return ''Error: N too large for D_max; no solution exists!''
}
where 3^h denotes a sequence of h 3's.
```

Algorithm 2: Finding an optimal B for Problem 2

larger, although not much. Here, we also see that the tail of our B may contain a single element of 5; this shows that recursion is indeed required for our algorithm to always find an optimal solution.

– In Set 2, the input contains relatively large primes. Buttyán's algorithm cannot improve upon the Classic tree at all, leaving much room for improvement by Algorithm 2. The difference in performance is about as large as between the Classic and Buttyán trees in Set 1.
– For Set 3, Buttyán's algorithm and Algorithm 2 perform similarly and yield the same output.

Table 1. Test cases

Input	Classic	Buttyán	Optimized Buttyán
Set 1[2]: $N = 27000$, $D_{max} = 90$	$(30, 30, 30)$ $R = 0.9355$	$(72, 5, 5, 5, 3)$ $R = 0.9725$	$(73, 5, 3, 3, 3, 3)$ $R = 0.9729 (N' = 29565)$
Set 2[3]: $N = 24389$, $D_{max} = 100$	$(29, 29, 29)$ $R = 0.9333$	$(29, 29, 29)$ $R = 0.9333$	$(84, 4, 3, 3, 3, 3)$, $R = 0.9764 (N' = 27216)$
Set 3[4]: $N = 1728$, $D_{max} = 36$	$(12, 12, 12)$ $R = 0.98462$	$(24, 4, 3, 3, 2)$ $R = 0.9194$	$(24, 4, 3, 3, 2)$ $R = 0.9194$

[2] Buttyán's example
[3] Example with prime numbers
[4] Relatively small tree; note that Buttyán's algorithm produces an optimal output in this case

3.1 Consequences of Larger Trees

Algorithm 2 can lead to trees that exceed the strictly required number of leaves (with $N' > N$). We argue that this has practical advantages, but should also be taken into account when judging the anonymity of such trees.

A larger tree will allow for addition of tags at a later time, which may be desirable in practice. Ideally, creating and balancing a tree should be done only once, and therefore the tree should accommodate all the tags *ever expected to enter the system*. In systems where growth is anticipated, having a larger tree that is ready for the future is good practice.

Also, since we are defending against tampering attacks, replacement of compromised tags should be taken into consideration. Replacement tags should contain *new key material*, lest they be reintroduced with keys that are already fully disclosed (immediately limiting their anonymity). Having unused leaves in the tree seems ideal for this purpose.

When choosing *which* leaves to actually use as tags (initially and for replacements), we suggest to select a sufficient number of branches at the level $d-1$ *at random*, and to randomly initialize tags from these branches. This to create a subtree of initialized tags that is as close to the original (optimal) shape as possible, without introducing order in the system which might be exploited.

Finally note that tags corresponding to unused leaves in the tree cannot be encountered by adversaries in the field. For this reason, they do not contribute to the size of the set among which targets need to be distinguished. This means that (some) anonymity sets will appear larger than they are in actuality. Because our anonymity measures are all based on set sizes, to prevent overestimating the results of our solution, we apply corrections as detailed below.

Theorem 4. *If N tags are placed uniformly at random in a tree with $\prod(B) = N' > N$, then the expected resistance to c member compromise (R_c) equals $\frac{N}{N'} R_c(B)$.*

Proof. Consider a particular choice of N tags within the set $\{t_i \mid 1 \leq i \leq N'\}$. Consider a particular choice of c compromised tags within the N tags. Let $1 \leq i \leq N'$. Then $P(t_i)$ denotes the anonymity set of tag t_i, considering only the N chosen tags. Note that $P(t_i)$ will be empty when tag t_i is not one of the N chosen tags. On the other hand, $P'(t_i)$ denotes the anonymity set of tag t_i, considering all N' elements.

It is clear that, when averaging over all possible choices of N tags, and all possible choices of c compromised tags,

$$E[|P'(t_i)|] = \frac{N'}{N} \cdot E[|P(t_i)|]$$

Therefore, $R_c = \bar{S}_{\langle - \rangle}(c)/N = E[\frac{1}{N} \sum_{i=1}^{N} \frac{|P(t_i)|}{N}]$, which equals $E[\frac{1}{N} \sum_{i=1}^{N'} \frac{|P(t_i)|}{N}]$, because $P(t_i)$ is empty when tag t_i was not chosen. Consequently, $R_c = E[\frac{1}{N} \sum_{i=1}^{N'} \frac{|P'(t_i)|}{N'}] = \frac{N}{N'} R_c(B)$. □

3.2 Simulation for $c > 1$

Section 3 has already shown that our proposal can yield lexicographically larger B than Buttyán's approach, and consequently better anonymity measures when $c = 1$. For $c > 1$, the theoretical analysis becomes very complex, so we compare our approach to Classic and Buttyán trees by means of computer simulations. Both Buttyán and our proposal assume a maximum allowed authentication delay, and try to provide optimal privacy within this boundary. Therefore, it is sensible to compare the *anonymity* of the two solutions, but not their efficiency (in terms of authentication delay), because both solutions are constructed based on the same maximum authentication delay D_{max}.

We will calculate $\bar{S}_0(c)$ and $\bar{S}_{\langle - \rangle}(c)$.Using code written in C++, we iterate over all possible scenarios in an efficient way, making use of the fact that many scenarios are equivalent with regard to set sizes. The minimum and (weighted) average of $\bar{S}$ taken over all these scenarios is stored as output. Because the number of scenarios grows rapidly as the number of compromised tags increases, we limit ourselves to cases with $1 \leq c \leq 100$.

For trees with $N' > N$, results are corrected by scaling down as discussed in Section 3.1.

Table 1 shows the three input sets for which we have evaluated the Classic, Buttyán and Optimized Buttyán trees. The following figures provide a graphical comparison of the performance of different trees under various conditions. A selection of the most relevant datasets was made, and some of the figures show partial graphs to provide the required level of detail. We will discuss how these results relate to our hypotheses and claims.

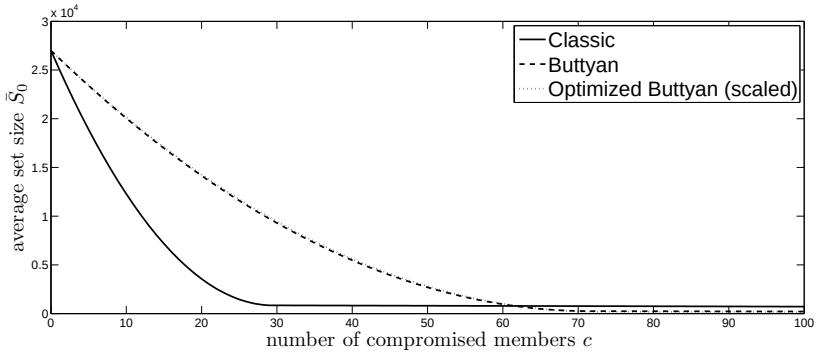

Fig. 3: $\bar{S}_0(c)$ for Set 1

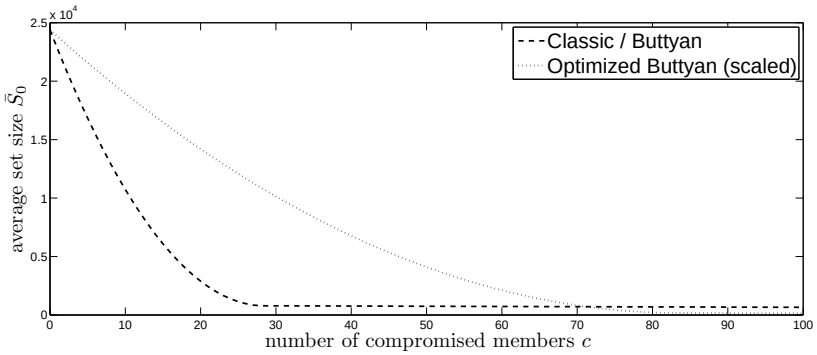

Fig. 4: $\bar{S}_0(c)$ for Set 2

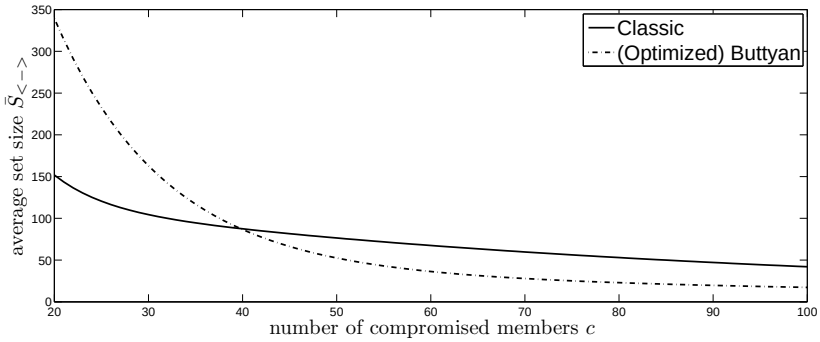

Fig. 5: $\bar{S}_{\langle - \rangle}(c)$ for Set 3

For Set 1, the Optimized Buttyán trees seemed to outperform the regular Buttyán tree significantly in terms of $\bar{S}_0(c)$. However, after correction (to account for the fact that $N' > N$), there is little difference in actual performance (Figure 3). The same trend was observed for $S_{\langle - \rangle}(c)$, and the result for Set 3 (not displayed here). We note that the performance of the Optimized Buttyán trees in no case drops below that of the original Buttyántrees.

Results for Set 2 differ, because the Buttyán tree there is strongly suboptimal in shape, with Figure 4 clearly showing the advantage of the Optimized Buttyán tree. This is due to the fact that the chosen N has too few prime factors for Buttyán's algorithm to work with. In similar cases, the same problem will occur to a greater or lesser extent.

Based on Figure 5, we observe a turning point where the classic tree starts to outperform the (Optimized) Buttyán trees. This occurs in all graphs, at $c = b_1$ for $\bar{S}_0$ and at $c \approx b_1$ for $S_{\langle - \rangle}$. At these points, the decrease of $\bar{S}$ slows causing the graph to *seemingly* settle into a steady minimum. We can explain this by the fact that at [around] this point, the last very large anonymity set has been

[is expected to have been] broken down, because each top-level branch contains [can be expected to contain] at least one compromised tag. Because subsequent compromised tags then fall into smaller sets, the adversary will learn little new information; he has obtained the most important keys in the tree already. In such a worrying scenario, what little anonymity tags have left depends upon the keys in lower branches. Classic trees retain slightly more anonymity, because they have larger branching factors at the bottom levels. However, given the (by then) minimal values of $\bar{S}$ overall, the *absolute* advantage is not large.

4 Conclusions and Future Work

Our proposed Algorithm 2 yields better results than Buttyán's original approach, when it comes to finding the lexicographically largest B. We have provided proof that the solution is optimal in terms of optimization problem 2. The output is at least as good (often superior, in some cases identical) to Buttyán trees in terms of R, $\bar{S}_0(c)$ and $\bar{S}_{\langle - \rangle}(c)$.

Algorithm 2 can result in trees with $N' \geq N$, which may be advantageous in growing systems or when replacing compromised tags. It also means that care must be taken not to overestimate the anonymity, leading us to apply corrections to our simulation results. The corrected results still clearly show the advantage of our tree construction to both Classic and Buttyán trees.

Future work:
We wish to expand our current results, by taking side-channel knowledge and adaptive adversaries into consideration. A paper detailing the results of this has recently been accepted at the SecureComm2011 Conference [3]. Defending against adaptive attacks leads to a different optimization problem (and a different optimal tree shape), and thus to a trade-off between defending against naive and adaptive adversaries. This is related to our observation that anonymity is mostly lost when $c \approx b_1$, and the fact that this process may be accelerated by adversaries who are able to accurately choose which tags they compromise.

As is evident from our examples, Algorithm 2 provides results that are much better, better, or identical to Buttyán's results, depending on the exact inputs N and D_{max}. Additional experimentation could show what the expected gain in anonymity is in the average case (for randomly chosen N and D_{max}), thus better illustrating the actual performance of Algorithm 2 in practice.

Finally, to get back to Buttyán's group-based proposal in [8], its suspected weakness lies in the fact that the final stage of narrowing down IDs is essentially skipped (the unique ID can be simply decrypted and read). Especially adaptive attackers that can choose their tags with some confidence, could very quickly remove all anonymity within the system by choosing one tag from each group. Tree-based systems still preserve some measure of anonymity in these cases. A formal analysis of this problem (both for naive and adaptive attacks) forms another direction for future work.

References

1. Gildas Avoine, Etienne Dysli, and Philippe Oechslin. Reducing Time Complexity in RFID Systems. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *LNCS*, pages 291–306, Kingston, Canada, August 2005. Springer-Verlag.
2. Gildas Avoine, Pascal Junod, and Philippe Oechslin. Time-Memory Trade-Offs: False Alarm Detection Using Checkpoints, Extended Version. Technical report, 2005. LASEC-REPORT-2005-002.
3. Michael Beye and Thijs Veugen. Anonymity for key-trees with adaptive adversaries, 2011. Accepted at SecureComm2011, 7th International ICST Conference on Security and Privacy in Communication Networks.
4. Levente Buttyán, Tamás Holczer, and István Vajda. Optimal Key-Trees for Tree-Based Private Authentication. In *In Proceedings of the International Workshop on Privacy Enhancing Technologies (PET)*, June 2006. Springer.
5. Ivan Damgård and Michael Østergaard Pedersen. RFID Security: Tradeoffs between Security and Efficiency. Cryptology ePrint Archive, Report 2006/234, 2006.
6. Claudia Díaz. Anonymity Metrics Revisited. In Shlomi Dolev, Rafail Ostrovsky, and Andreas Pfitzmann, editors, *Anonymous Communication and its Applications*, number 05411 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.
7. Hannes Federrath, editor. *Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology*, volume 2009 of *LNCS*. Springer-Verlag, 2001.
8. Tamas Holczer Istvan Vajda Gildas Avoine, Levente Buttyán. Group-based private authentication. In *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6, 2007.
9. M. Hellman. A cryptanalytic time-memory trade-off. In *Information Theory, IEEE Transactions on*, volume 26, pages 401–406, July 1980.
10. Ari Juels and Stephen A. Weis. Defining Strong Privacy for RFID. In *PERCOMW '07: Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 342–347, Washington, DC, USA, 2007. IEEE Computer Society.
11. David Molnar and David Wagner. Privacy and security in library RFID: issues, practices, and architectures. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 210–219, New York, NY, USA, 2004. ACM.
12. Yasunobu Nohara, Toru Nakamura, Kensuke Baba, Sozo Inoue, and Hiroto Yasuura. Unlinkable identification for large-scale rfid systems. *Information and Media Technologies*, 1(2):1182–1190, 2006.
13. Karsten Nohl and David Evans. Quantifying information leakage in tree-based hash protocols (short paper). In Peng Ning, Sihan Qing, and Ninghui Li, editors, *ICICS*, volume 4307 of *LNCS*, pages 228–237. Springer, 2006.
14. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cryptographic Approach to "Privacy-Friendly" Tags. In *RFID Privacy Workshop*, MIT, MA, USA, November 2003.
15. Stephen A. Weis, Sanjay E. Sarma, Ronald L. Rivest, and Daniel W. Engels. Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In Dieter Hutter, Günter Müller, Werner Stephan, and Markus Ullmann, editors, *SPC*, volume 2802 of *LNCS*, pages 201–212. Springer, 2003.

16. Sang-Soo Yeo and Sung Kwon Kim. Scalable and Flexible Privacy Protection Scheme for RFID Systems. In Refik Molva, Gene Tsudik, and Dirk Westhoff, editors, *European Workshop on Security and Privacy in Ad hoc and Sensor Networks – ESAS'05*, volume 3813 of *LNCS*, pages 153–163, Visegrad, Hungary, July 2005. Springer-Verlag.

A Proof of Theorem 2

The first observation is that for an optimal B, $\sum(B) = D_{max}$, otherwise $D_{max} - \sum(B)$ could be added to any element of B without violating the constraints while increasing $R(B)$. So we assume $\sum(B) = D_{max}$ in the proof, which uses four Lemmas, similar to the Lemmas of Buttyán's work [4]. It's also clear that an optimal B will have branching factors at least 2. The first Lemma, Lemma 1, shows that a branching vector can always be improved by ordering its elements in decreasing order. Lemma 3, using some bounds from Lemma 2, shows that given two branching factor vectors, the one with the larger first element is always at least as good as the other. Lemma 4 generalizes Lemma 3 by stating that given two branching factor vectors the first j elements of which are equal, the vector with the larger $(j+1)$-st element is always at least as good as the other.

These Lemma's together show that a lexicographically larger branching factor vector will always be at least as good as the lexicographically smaller branching factor vector (in case $\sum(B) = D_{max}$), so indeed the solution with maximal $R(B)$ to Problem 2 is achieved by the lexicographically largest vector that satisfies the constraints.

Lemma 1. *Let B be a branching factor vector, and let B^* be the vector that consists of the sorted permutation of the elements of B in decreasing order. If B satisfies the constraints of Problem 2, then B^* satisfies them too, and $R(B^*) \geq R(B)$.*

Proof. Since $\prod(B)$ is not altered by the permutation, we can refer to Buttyán's proof [4] of Lemma 1. □

Lemma 2. *Let $B = (b_1, \ldots, b_d)$ be a sorted branching vector (i.e. $b_1 \geq b_2 \geq \ldots \geq b_d$). We can give the following lower and upper bounds on $R(B)$:*

$$\left(1 - \frac{1}{b_1}\right)^2 \leq R(B) \leq R(b_1) = \frac{1 + (b_1 - 1)^2}{b_1^2}$$

Proof. The lower bound is identical to Buttyán, hence the proof [4] is as well. The upper bound is an improvement w.r.t. Buttyán, and is proven as follows. Let $M = \prod(B)$, then $\prod(B \backslash b_d) = M/b_d$. We derive for $d > 1$:

$$R(B) = \frac{1}{M^2}\left(1 + (b_d - 1)^2 + \sum_{i=1}^{d-1}(b_i - 1)^2 \prod_{j=i+1}^{d} b_j^2\right)$$

$$
\begin{aligned}
&= \frac{1}{M^2}\left(1 + (b_d - 1)^2 + \sum_{i=1}^{d-2}(b_i - 1)^2 \prod_{j=i+1}^{d} b_j^2 + (b_{d-1} - 1)^2 b_d^2\right) \\
&= R(B\backslash b_d) - \frac{b_d^2}{M^2}\left(1 + (b_{d-1} - 1)^2\right) + \frac{1}{M^2}\left(1 + (b_d - 1)^2 + (b_{d-1} - 1)^2 b_d^2\right) \\
&= R(B\backslash b_d) + \frac{2 - 2b_d}{M^2} \\
&< R(B\backslash b_d)
\end{aligned}
$$

and by recursively applying this inequality also $R(B) \leq R(b_1)$. □

Lemma 3. *Let $B = (b_1, \ldots, b_d)$ and $B' = (b'_1, \ldots, b'_{d'})$ be two sorted branching factor vectors (i.e. $b_1 \geq b_2 \geq \ldots \geq b_d$, $b'_1 \geq b'_2 \geq \ldots \geq b'_{d'}$) that satisfy the constraints of Problem 2. Then, $b_1 > b'_1$ implies $R(B) \geq R(B')$.*

Proof. We first prove the statement for $b'_1 \geq 3$. From Lemma 2 we know that

$$R(B') \leq \frac{1 + (b'_1 - 1)^2}{{b'_1}^2}$$

and

$$R(B) \geq \left(1 - \frac{1}{b_1}\right)^2 > \left(1 - \frac{1}{b'_1 + 1}\right)^2$$

which follows from the fact that $b_1 > b'_1$. A straightforward calculation shows that $(1 - \frac{1}{b'_1+1})^2 \geq \frac{1+(b'_1-1)^2}{{b'_1}^2}$ whenever $b'_1 \geq 3$, and thus $R(B) \geq R(B')$.

So the remaining case is $b'_1 = 2$. Since B' is ordered, each element of B' will equal 2. If $d' = 1$ then by our previous assumption $D_{max} = \sum(B') = 2$, but this contradicts $D_{max} = \sum(B) \geq 3$, so we know $d' \geq 2$. The resistance $R(B')$ is readily computed as $R(B') = \frac{1}{3}(2 \cdot 4^{-d} + 1)$, which will be at most $\frac{3}{8}$ (when $d' = 2$). Since $R(B) \geq (1 - \frac{1}{b_1})^2 > (1 - \frac{1}{3})^2 = \frac{4}{9}$, it follows that also in this case $R(B) \geq R(B')$. □

Lemma 4. *Let $B = (b_1, \ldots, b_d)$ and $B' = (b'_1, \ldots, b'_{d'})$ be two sorted branching factor vectors (i.e. $b_1 \geq b_2 \geq \ldots \geq b_d$, $b'_1 \geq b'_2 \geq \ldots \geq b'_{d'}$) that satisfy the constraints of Problem 2. Let j, $1 \leq j < \min(d, d')$, be such that $b_i = b'_i$ for all i, $1 \leq i \leq j$, and $b_{j+1} > b'_{j+1}$, then $R(B) \geq R(B')$.*

Proof. It is easy to show that $R(B) = \left(\frac{b_1-1}{b_1}\right)^2 + \frac{1}{b_1^2} \cdot R(B\backslash b_1)$. Therefore, since $b_1 = b'_1$, $R(B) \geq R(B')$ whenever $R(B\backslash b_1) \geq R(B'\backslash b'_1)$. By recursively applying this rule, and using Lemma 3, which shows that $R(B\backslash\{b_1, \ldots, b_j\}) \geq R(B'\backslash\{b'_1, \ldots, b'_j\})$, the proof is complete. The proof is identical to the proof of Buttyán's Lemma 4 [4]. □