Key Recovery Using Noised Secret Sharing with Discounts Over Large Clouds

Sushil Jajodia
George Mason University
Fairfax, VA, USA
{jajodia@gmu.edu}

Thomas Schwarz,
Universidad Católica del Uruguay
Montevideo, Uruguay
{tschwarz@ucu.edu.uy}

Witold Litwin
Lamsade, Université Paris Dauphine
Paris, France
{witold.litwin@dauphine.fr}

Abstract—**Encryption key loss problem is Achilles's heel of cryptography. Key escrow helps, but favors disclosures. Schemes for recoverable encryption keys through noised secret sharing alleviate the dilemma. Key owner escrows a specifically encrypted backup. The recovery needs a large cloud. Cloud cost, money trail… should rarefy illegal attempts. We now propose noised secret sharing schemes supporting discounts. The recovery request with discount code lowers the recovery complexity, easily by orders of magnitude. A smaller cloud may suffice for the same recovery timing. Alternatively, same cloud may provide faster recovery etc. Our schemes appear useful for users attracted to Big Data, but afraid of possibly humongous consequences of the key loss or data disclosure.**

Keywords-clouds; big data; privacy; key recovery.

I. INTRODUCTION

Key recovery is a classical goal. Key escrow, i.e., entrusting a key copy with some (escrow) agent, was proposed as a basic solution. The idea did not catch. Key owners seem fearing the key disclosure, as source of irresistible temptations for some. More complex key escrow schemes, e.g., with recovery rights verification or through secondary encryption of the copy with the key entrusted to another party, are almost no noticeable in practice. See the related work section in [3]. Not having key escrow, on the other hand, exposes the data to the key loss, especially if the owner disappears with. Modern encryption schemes, e.g., AES, make data unrecoverable then.

The fear of key loss particularly concerns many users attracted to Big Data idea. It is the common knowledge that safety and efficiency of related manipulations require the data outsourcing to some cloud. However, one knows well that most users are reluctant to outsource data in clear, [1]. Many can't do it simply by law. Many pro may accept therefore the idea only if they may encrypt the outsourcing. A homomorphic code allows in particular for the arithmetic calculations over the encrypted data (directly) in the cloud, [6]. This is a strong need, as many Big Data queries use value expressions, distributed in addition for efficient evaluation using Map-Reduce, [5]. Even with all these tools, many or most of these users remain deterred up to now. Big Data may render indeed key loss consequences accordingly huge. It could worth years of work of many people. Vice versa, a misuse of an escrowed key copy may result in the cloud content disclosure. This may lead to consequences of scale equally calamitous.

Schemes for *recoverable encryption through noised secret sharing*, RE_{NS} schemes in short, appeared as a new solution to the dilemma, [2] and [3]. The key owner escrows a specifically encrypted backup. The brute-force key recovery, from the backup alone, is always feasible although intentionally hard. Its complexity, as measured by the number of instructions the recovery may need, is arbitrarily fixed by the owner, depending on the trust in the escrow agent. As the result, the recovery timing on the escrow's site (node) alone, should become impractical, e.g., should last dozens of days at least. This timing results from some key-owner defined integer M, called *backup (encryption) hardness*, providing $O(M)$ worst-case complexity. Nevertheless, the actual maximal recovery time desired by the recovery requestor remains practical. The recovery uses a sufficiently large N-node cloud, providing about linear $O(M/N)$ recovery speed-up. Practical timing, e.g., in minutes, is expected to imply N in thousands. The escrow is not expected to maintain such a large cloud on premises. Hiring it from some external provider is then is a necessity. That one should be usually somehow costly and easily noticeable through numerous logs, money trail…. One may expect illegal disclosure attempts, e.g., by an escrow side insider, to rarefy. They may be

978-0-7695-5137-1/13 $26.00 © 2013 IEEE
DOI 10.1109/SocialCom.2013.105

expected much less tempting than from a simple key copy.

We now extend known RE_{NS} schemes with the concept of *discount*. As the name suggests, a discount results from a code that lowers the recovery cost with respect to the brute-force one. The reduction is easily by orders of magnitude as it will appear. We speak then about the *discounted recovery*. We furthermore rather designate the brute-force one from now on as the *full-cost* one. Technically, the code lowers the recovery complexity for both the worst and the average cases.

The code is an m-bit string; $m = 0,1\ldots$. The $m = 0$ tacitly means the no-discount request, i.e., the full-cost one. Otherwise, we expect $m = 8$ or $m = 16$ at most, in practice. With respect to the full-cost recovery, such codes lower both complexities, respectively 256 for $m = 8$ and 64K times for $m = 16$. The minimal actual discount is for $m = 1$ and is 50%. Accordingly, the discount may greatly reduce recovery timing and/or the cloud size and *au finale* the cloud cost for the requestor.

The code may appear to the key owner in some convenient form. The minimal discount requires only retaining that the code is even or odd. Otherwise, one choice may be a single 16b Unicode digit. Or, it can be one or two (extended) 8b ASCII digits. Alternatively, one may choose $2 \div 4$ hex digits, etc. One expects such codes easy to retain, e.g., on a smartphone, or simply in memory. Especially, it should be very easy for the minimal discount. Recall that Europeans are routinely trained to keep in mind their 4-digit credit card codes. They are strongly advised not to store them anywhere, (especially on the credit cards themselves).

The requestor sends the discount code to the escrow within the (discounted) recovery request. The code amends the processing of the otherwise always feasible full-cost recovery. For any given key, only specific codes lead to a discounted recovery. Any discount provided triggers nevertheless a recovery attempt with the associated cost. An unsuccessful attempt also respects the requestor's timeline. It doubles however the average cost of the successful one. Finally, every code is granted successful only for one key. For any other one, it basically acts as (purely) random guess of what should be the actual one. With costly consequences, we just discussed. All together tampering with a discount code should be infrequent.

Globally, we show below that for all these reasons discounts appear a highly useful capability for an RE_{NS} scheme. The existing 2-share schemes generalize easily. The key remains "never-lost", with (illegal) disclosure cumbersome at will for the attacker. Yet, the legal recovery by the discount possessor remains cheap and thus practical. This combination is unique up to now for a key recovery scheme. Especially, it can greatly help the already mentioned potential Big Data users.

Below, we first define and analyze the RE_{NS} schemes with discounts using, so-called 2-share noised secret sharing. We generalize for this purpose the two related schemes defined in [2]. These are told respectively *static* and *scalable* and are analyzed further more in depth in [3]. We define the backup and the discount creation, then the discount-based recovery calculation. Next, we analyze the correctness, the complexity and the safety of the resulting schemes. Afterwards, we generalize the 2-share recovery calculation to a $(k + 1)$ -share one with $k > 1$ at key owner will. We show attractive property of such schemes. Next, we briefly address the related work and we finally conclude. Space limitation forced us to evacuate all the figures we discuss into [6].

II. BACKUP CREATION

Let K be the key to backup, e.g. a 256b long AES key. The key owner or rather the owner's client program, running on owner's site, say O in every case, first creates a usual 2-share secret, with shares, say s_0 and $s_1 = K$ XOR s_0, Fig. 1a. It is the common knowledge that $K = s_0$ XOR s_1. Next, O chooses some time D, e.g., 70 days. D is the intended recovery time at the escrow's site alone, assuming it a 1-node (core) configuration. The choice of D value reflects O's trust in the escrow service that that no (illegal) disclosure attempt occurs there. Lower it is, higher D should be.

After that, Fig. 1b, C defines the *hint* $h = H(s_0)$, using some one-way hash function H, e.g., SHA_{256}. We recall that in practice (i) h is unique for any s_0 and (ii) it is impossible for good H such as the one mentioned, to calculate s_0 as $H^{-1}(h)$. Next, C determines the backup hardness M. This parameter is the maximal number of *match attempts* $H(s) =? h$ where each s is a different integer that could be s_0, sufficient to find the successful match. Also, M is the owner's expectation of the number of match attempts that 1-node site may perform at most in time D. Next, if there is no $g = 1, 2\ldots$ such that $M = 2^g$, then C verifies whether the $\lceil \log_2 M \rceil$ - bit long suffix r of s_0 is $r < M$. If not, then C chooses random $q \in I = [0, M[$ and sets up $r := q$, i.e., substitutes q as new suffix r of s_0, and recalculates both shares accordingly. Next, C calculates the bit-length of r, i.e., calculates an integer g such that either $g = \log_2 M$ when such value exists, or $g = \lceil \log_2 M \rceil$. Then, C cuts off r from s_0, i.e., produces the integer $p = s_0 \setminus 2^g$, where p denotes thus in fact the remaining prefix of $s_0 = p \,|r$.

As in [2], we call below each x a *noise* and I is the *noise space*. Then, $f = p|0\ldots0$ is the *base noise* share, while s_0 a *noised* one. The naming comes from the backup representation of s_0 that is $P = (p, h, M)$. P makes s_0 hidden somewhere among M different *noise* shares, formed each as $s = f + x$; $x = 0,1..M-1$, Fig.

1b. The noise shares form the *noise share space,* say Q, with card $|Q|$ = card $|I| = M$. Each s is an integer, as we just said and can be s_0. This happens iff $x = r$. The only known way to find out is to attempt the match. By the well-known properties of a good 1-way hash, this one succeeds iff $s = s_0$. The backup sent to the escrow is the couple (s_1, P).

Notice that, while the backup creation is quite similar to that in [2], the definition of noise shares differs. The rationale (that we do not plan to address further here) is a programmatically simpler discount calculation.

III. DISCOUNT DEFINITION

In [2], the full-cost recovery, i.e., using exclusively the backup as above defined, was the only capability of RE_{NS} schemes defined there. One may nevertheless observe that the requestor could also forward with the backup request some prior knowledge of s_0 that could lower the recovery complexity. For instance, the key owner could observe that s_0 is an odd integer. This would lower the complexity by 50%, as we show. We say that any such knowledge defines a *discount*, of 50% in this case.

More precisely, the key owner defines the discount for a given backup according to an RE_{NS} scheme, by choosing some *discount code*. For what follows, the code is simply an m-bit suffix of the noised share s_0 ; $m = 0,1….$Fig. 2. The value of $m = 0$ tacitly means the no-discount request, i.e., the full-cost one. Otherwise, as we signaled already, we will talk about a *discounted* recovery. We expect for the latter $m = 8$ or $m = 16$ at most in practice. We call *discount value* the complexity reduction that the recovery with the code offers with respect to the full-cost. The analysis later on shows that the value of any m-bit long code is 2^m for both, worst case and average complexities.

The reason for such value is the 2^m times smaller *discounted (backup) hardness, i.e.,* the accordingly smaller maximal number of match attempts towards the successful one for sure. That discount with respect to M characterizing the full-cost recovery, is due to smaller noise space I'. That one becomes of size $|I'| = M' = M/2^m$, Fig. 2. We will show it in the sections that follow. Notice nevertheless already that the suffix r must be $r = r'|d$ for some noise $r' \in I'$. We expect accordingly in practice the reduction of the worst case, as well as of the average, complexity of up to $2^8 = 256$ for $m = 8$ and $2^{16} = 64$K times for $m = 16$.

Discount codes that short may appear to the key owner as a single 16b Unicode digit or as one or two (extended) 8b ASCII digits, or as $2 \div 4$ hex digits. One may expect them generally easy to retain, e.g., on a smartphone, or just in memory. Recall that Europeans are routinely trained to keep in mind their 4-digit credit card codes.

As the result that will appear, these codes may lower the recovery time $2^{8 \div 16}$ times with respect to the full-cost timing for the same cloud size. Alternatively, they may reduce the cloud size by the same value, while keeping the same timing. The discounted hardness also allows combining both reductions. *Au finale*, the discount decreases the cloud cost. Obviously, the necessary condition for a successful match attempt is that the noise share embeds the code provided. The rationale for the discounted recovery algorithm we define in next section is to attempt matches only for such shares.

Ex. 1. Key owner's client key encryption generates s_0 = '….0000 0000 0110 1010'. In Unicode the owner sees the 16b suffix above, qualifying for the discount code, as a single symbol 'j', hence we have : s_0 = '….j'. In extended ASCII, the bits appear as two characters s_0 = '….∅j', where ∅ denotes the NULL, i.e., '00' character. Finally, the owner observes that s_0 is an even integer. The owner retains as discount code representation for storage somewhere or simply for memory, the Unicode representation, i.e., 'j'. Later, this same single character may represent either discount code: the 8b-long d = '0110 1010' in ASCII and the 16b-long d = '0000 0000 0110 1010'. The owner may decide only when needed which discount to choose. The former will offer the discount value of 256 times, the latter will provide 64K times hardness reduction. As 2nd line protection against loosing even this simple code value, the owner retains that s_0 is even. At the minimum, the code d = '0' will still provide 50% discount, as it will appear. In the very last but not least, resort, the full-cost recovery is always feasible.

RECOVERY

A. Recovery Request

The escrow performs the recovery upon the legitimate request. How the escrow knows which request is legitimate is out of scope here. Recovery schemes with discount discussed below reuse the scheme for full-cost only recovery defined in [2] and [3]. The recovery request has in particular the same form, augmented however with the discount code. It is thus formally the tuple $P_d = (P, R, d)$. As for the full-cost only recovery request in [2], here, R designates the desired maximal recovery time, e.g., 10 min. Recovery schemes in [2], as well as those below, consider then R as the upper bound on recovery computation time over any cloud node used. They thus fulfill the user's desire for sure provided the cloud overhead consisting of messaging, node allocation etc. times, is negligible.

B. Full-Cost Recovery

If the request has no discount, i.e., $m = 0$ in d, the escrow proceeds with the *full-cost* recovery. The schemes in [2] apply then as they are, except for the

revised s_0 base noise share definitions. The escrow forwards thus P_d to some cloud node C, called *coordinator*, with the exception of s_1. In this way no cloud insider can disclose the recovered key.

With respect to the actual execution on the cloud, managed by C, we recall now that [2] defines two basic schemes. We called them respectively *static* and *scalable* partitioning. The former was proposed for a homogenous cloud. The latter targets a heterogeneous one. Their common characteristic is that the recovery calculations attempt the matches over different noise shares $f + m$, until the successful match. This one must occur, but attempts may possibly explore even every m in I. Both schemes partition the attempts over N nodes, with the linear speed-up $O(N)$. The choice of N value depends on the scheme. In both cases, it makes the recovery computation at each node fitting the time bound provided by the requestor, e.g., 10 mins. As the result, the whole calculation fits this bound. Typically, N should be possibly in thousands, as we discussed.

The cloud delivers the noised share s_0 found to the escrow. The escrow XORs it with s_1 and, finally, delivers the key to the requestor.

C. Discounted Recovery

The discounted recovery request differs from the full-cost one by additional presence of the discount code with $m > 0$. The cloud uses then the *discounted recovery scheme* that follows. Its rationale is that the noised share has to have d as m-bit long suffix, Fig. 2. The only noise shares in the noise share space that could match must have the same suffix. The recovery processing should generate all and only such shares. We say they form the *reduced* noise share space. The prefixes of noises in these shares, preceding suffix d in each noise, must form a subspace I' with noises $x = 0,1,\ldots M' - 1$ where M' is M with d cut off from its binary representation. We call it *reduced* noise space. One may explore only this space. The exploration should form for each visited noise x, the noise $x|d$. It then should concatenate it then with prefix p of s_0 to form noise share $p|x|d$, belonging to the reduced noise share space, for the match attempt. The successful match occurs when for some x, $H(p|x|d) = H(p|r'|d) = h$. See Fig. 2 for $d =$ 'j'. The exploration of I' may use either partitioning scheme for full-cost recovery in [2] or [3]. The details of the scheme we sketch now follow these considerations. We explain it more and finalize its correctness proof afterwards.

1) C calculates $M' = M \setminus 2^m$.

2) For $m > 0$, Step (1) defines the reduced noise space $I' = [0, M'[$ we spoke about. C initiates the static or the scalable scheme for this space, i.e., uses M' instead of M. We recall that this step determines later N, in function of M' and R.

3) C delivers to each of N nodes the "usual" full-cost request for match attempts and the discount code d. The delivery is direct for the static scheme, and may be indirect for the scalable one, [2]. The scalable scheme determines N progressively, while propagating the request for match attempts.

4) Each node n; with $n = 0,1\ldots N - 1$ for the static scheme and perhaps noncontiguous integers for n in some $[0\ldots N'[$ where $N' \geq N$, for the scalable one, one first calculates the *base share* f' for the discounted recovery. According to what we have said, we define it as the smallest possible with the suffix d, i.e. we clearly have $f' = `p|0' + d$. Notice that f' generalizes f for full-cost recovery, since $f' = f$ for $m = 0$.

5) Next, using M' instead of M, every node calculates one after another every value of noise x for which it should generate noise share s for match attempt $H(s) =? h$. For each x used, each node calculates s as $s = f' + x * 2^m$. The noises used at each node n depend on n and on the distribution scheme used, in the same way as for the full-cost recovery. Fig. 3 illustrates the issue that we address also in depth later in this section.

6) As for the full-cost recovery, every node attempts the match for each s. If the match occurs, the node reports s as the noised share s_0 to C, unless the node is C itself. The node terminates the service then, freeing all the resources.

7) Otherwise, the node continues the attempts. It does so until the last relevant x or until the node receives the *termination* message from C. This one requests the node to terminate, i.e., to stop the service and free all the resources.

8) Assuming the cloud finds in this way s_0, C returns it to the escrow. The escrow XORs it with s_1 and returns the recovered key to the requestor. C sends also out to the already mention termination message. For the static scheme it may send it simply directly to every node. It may alternatively send indirectly to most of the nodes, through the direct send-out to a few selected ones only that propagate it further in parallel. For the scalable scheme, the latter strategy is usually the only possibility. Space limits prevent dealing with more details.

9) For $m = 0$, i.e., the full-cost recovery, C must get s_0 or there is the failure of a cloud node or of the network connection between C and that node. One can reasonably expect such a failure to be very rare. We thus avoid discussing here the related details. Rules indicated for schemes without discount applies fully, besides.

10) For $m > 0$ in contrast, if C does not get s_0 a new cause may be the *invalid d* that is different from the actual one in s_0. The legitimate requestor made perhaps an error, or the discount came from an intruder… As before, the cause may be also a cloud failure, as for $m = 0$. C cannot distinguish from the above scheme between the cases. C acts then as if d

was invalid. The rationale is that this cause may be expected orders of magnitude more likely than the others. *C* reports to the escrow accordingly. It then terminates as discussed in Step 8.

11) As the result, we expect the requestor to usually send a different *d*. The whole algorithm restarts. Very rarely, the requestor may confirm nevertheless *d* as valid. *C* starts then a specific procedure. That one also restarts the recovery, but with additional features. These discriminate for sure at termination time whether there is a failure or *d* is invalid. The former case can still "hide" an invalid *d*. In the latter case, *C* reports to the escrow again that informs the recovery requestor accordingly. There are various ways to design that procedure. All should be nevertheless more complex, hence more expensive, than the basic one above. Again, we cannot address the related details here.

Discussion. Fig. 3 and Ex. 2 below illustrate the discounted recovery algorithm. The figure shows the distributed partitioned noise and noise share spaces over *N* nodes, for the discounted recovery using the static scheme and code 'j' from Ex. 1. The total size of the noise space and that of the noise share space is now *M'*. As for *M* for the full-cost recovery in [2], but for *M'* here, a noise subspace on node *n* ; $n = 0,1\ldots N-1$; contains each and every noise $x < M'$ and such that $x \bmod N = n$. The size of each noise subspace is $M' \setminus N$ or is $M' \setminus N + 1$. The sizes of noise share subspaces at each node are accordingly the same.

Fig. 3 and Ex. 2 illustrate the discounted recovery algorithm. The figure shows the distributed partitioned noise and noise share spaces over *N* nodes, for the discounted recovery using the static scheme and code 'j' from Ex. 1. The total size of the noise space and that of the noise share space is now *M'*. As for *M* for the full-cost recovery in [2], but for *M'* here, a noise subspace on node *n* ; $n = 0,1\ldots N-1$; contains each and every noise $x < M'$ and such that $x \bmod N = n$. The size of each noise subspace is $M' \setminus N$ or is $M' \setminus N + 1$. The sizes of noise share subspaces at each node are accordingly the same.

These sizes are in practice 2^m times smaller for $m > 0$ than for *M* and the same *N* in [2]. It is the same for *M'* with the same *N* and $m = 0$ and in the algorithm above, generating then the full-cost recovery. At each node, the discounted recovery has accordingly 2^m times less match attempts to perform at worst for any $m > 0$, than for $m = 0$. This property leads to new possibilities for the recovery requestor aiming at best advantage of a discount. We address these issues in Section 5 below.

Ex. 2. Consider $M = 2^{50}$ which should be rather typical. Suppose the noise shares 256b long, as an AES key. We have the base share (for the full-cost recovery) f = 'p |0….0000' with some prefix p and zero value suffix over 50 bits. Suppose further $m = 2$ and d = '01'. *C* calculates *M'* as $M = 2^{48}$. The base noise share for the discounted recovery is f' = 'p |0…00000000000|01'. Suppose the use of the static scheme and that after the calculations for *M'* as in [2] for *M*, we have *N* = 1K, hence $n = 0,1\ldots1023$. Node 0 attempts the matches for noises $x = 0$, 1024, 2048…, i.e., with each successive *x* such that $x \bmod N = 0$ and till the largest such $x < M'$. Each *x* is multiplied by 2^2 then added to f', then node 0 attempts the match of the resulting noise share, etc.

In particular, node 0 always starts with the match attempt for $s = f'$. If no success, next attempt is for $s = f' + 1024*4$, hence s = 'p |0…010000000000|01'. Then, if needed, there is the attempt for $s = f' + 2048*4$, i.e., s = 'p |0…100000000000|01' etc. Likewise, node 1, attempts the match for noise $x = 1$. Then, may continue for $x = 1025, 2045\ldots$, i.e., where $x \bmod N = 1$ and till the largest such $x <$ M'. Node 1 starts thus with the attempt for $s = f' + 1*4$, i.e., s = 'p |0…00000000001|01'. Perhaps continues then for $s := s + 1024*4$, i.e., $s = f' + 1025*4$, that is for s = 'p |0…10000000001|01' etc. In general, as on Fig. 3, every node *n* attempts in this way the matches for each and only $x < M'$ that yields $x \bmod N = n$. The scalable partitioning has a more complex rule, see, e.g., [2] for it.

With respect to Steps 10 & 11, the rationale for acting first as if *d* was invalid is easy to see. Assuming the use of a 1K-node cloud and the cloud sufficiently reliable to make a double failure among these nodes unlikely, the probability that a failure strikes just the node that should find s_0 should normally be 1/1K. Everyone's experience with PINs of credit cards, passwords…, shows errors once every relatively few uses, perhaps once every couple of dozens at most. The invalid *d* case should thus happen dozens of times more often.

Next, observe that until the procedure in Step 11, the cloud acts in the same way for valid and invalid *d*. There is no way for *C* to distinguish between both upfront. An invalid *d* will thus cost the requestor more. This feature is intentional, expected to curb the discount tampering.

The integer division '\' by 2^m amounts to *m*-bit right shift. Likewise, the multiplication by 2^m performs the *m*-bit left shift. Dedicated shift functions may be faster than the arithmetic calculations. There are thus various ways to implement the algorithm we do not address further here.

IV. ALGORITHM ANALYSIS

A. Correctness

Basically, it should appear that for every *I,* every *f* and every *d*, each RE_{NS} schema under consideration generates for every *N* the match attempts for all and only noise shares ending with *d*, in the noise share space generated by noise space *I'*. Also, no such share should be generated twice. Finally, it should appear that the recovery always terminates.

Proof. We skip the last part as quite obvious, at least in the absence of failures, here. The proof of the rest is rather easy to see from the figures and Ex. 2. We also skip the tedious details, referring the reader to [2], especially for the scalable scheme. The calculus of *M'* obviously calculates the number of noises in *I* that terminates with *d*. This is the size of *I'* hence of *Q'*. In Step (4), each node calculates *f'* in the way that yields an integer being a noise share and such that (i) it ends with *d* and (ii) is greater than or equal to *f*. By definition this is *f'*. The loop at each node then attempts the matches using every noise *x* handled by the node. It should be clear from Fig. 3 and the example that whatever is then *N* and *d*, all the *M'* noises in *I'* are possibly explored and only once per noise. Hence are all the noise shares in *Q'* and *H* cannot map a share beyond *Q'*. The calculation of *s* in Step (4) produces clearly for each *x* the noise share ending with *d*. There cannot be such a share in the noise share space *I* (as well as in *I'*) missing from the distributed calculus. Also, one easily sees from Fig. 3 that no noise shares in *Q'* may be generated twice, whatever are the parameters there. Similar analysis holds for any partitioning that could be generated by the scalable scheme.

Finally, it is easy to see that the *termination protocol*, i.e., the rules for cloud service termination without a cloud failure, is also correct. Every node starting the attempts indeed terminates. It either gets the termination message from *C* or terminates itself after all possible attempts. Next, *C* also terminates either by providing s_0 or by finding an invalid *d*, as we discussed in Step 10. Finally, the recovery cannot produce in practice s_0 for an invalid *d*. We prove this point in Section 4.3 below, as it rather concerns the safety.

B. Complexity

An *m*-bit long *d* decreases the recovery calculation complexity (hardness) 2^m times in practice. Respectively, we have $O(M/2^m)$ for the worst case and $O(M/2^{m+1})$ on the average.

Proof. For the full-cost recovery, the complexity could be measured basically by the number of noises to try out: at most or on the average. Each noise may indeed trigger a match attempt. The computational cost of SHA256, as well as any other known good 1-way hash function dominated additional operations required, at the start-up or termination etc. of the algorithms. We had thus basically the complexity of $O(M)$ in the worst case, for both static and scalable schemes. For both schemes, the discounted recovery has at most *M'* noises to try out. This is 2^m time smaller. On the other hand, the discounted recovery algorithm requires an additional initial calculation of *M'*. Next, it requires the calculation of *f'*. Finally, at each attempt, there is an additional multiplication by 2^m. However, it is the common knowledge that the cumulated computational cost of a few such operations is again negligible with respect to that of SHA256 or another good 1-way hash calculation. Hence, we have basically the $O(M')$ worst case complexity, i.e., the $O(M/2^m)$ one.

For the average case, we had under similar assumptions $O(M/2)$ for the full-cost recovery. The reason was that both schemes enumerated all attempts till the successful one, while every noise, hence every noise share tried out, were equally likely to try out and succeed, provided a good 1-way hash, as we supposed. For the discounted recovery, every attempt uses again a different noise and at worst all noises *M'* noises are explored. The discount code is (pseudo)random, hence every code is equally likely. Also, the rest of s_0, beyond the discount code, is (pseudo)random. Hence, every noise share generated is again equally likely to be the noised one, under the same good 1-way hash assumption. We thus have on the average the $O(M'/2)$ complexity, hence $O(M/2^{m+1})$.

Ex. 3 Consider the running example in [2] where the encryption complexity is set up so that 1-node recovery would require up to prohibitive 70-days and 35 days on the average. To recover the key in 10 min at most instead, using the full-cost, a 10K-node wide cloud may do. The actual cost could be 200$. Consider that the owner retained our 8b discount code 'j', as in Ex. 1 and Fig. 3 previously discussed. Now, 40-node cloud may suffice for the same timing. Alternatively, the same 10K cloud, delivers the discounted recovery in up to a couple of seconds. In both cases, the cost theoretically drops to less than 1$. A 16b discount 'j' would lower these figs accordingly further. The requestor could even recover the key at her/his own presumably single node, in about 2m1ins.

C. Safety

1) Knowledge of a discount code cannot lower the complexity of the requested backup under values $O(M')$ at worst and $O(M'/2)$ on the average, provided by our algorithm (see below).

Proof. Our algorithm enumerates all attempts till the successful one (if any). Every attempt uses a different noise among *M'* and, at worst, all noises *M'* noises are explored. The rest of s_0, beyond the discount code, is (pseudo)random and thus independent of the discount code value. Also, for a good 1-way hash as we suppose, each such value is equally likely to generate the matching *f'*. Hence, whatever is a given a discount code, one cannot calculate from it or otherwise any *f'* that could be less or more likely than any other possible. No method exists that would allow to attack the requested backup from its given discount, towards lowering the complexity under that of our algorithm.

2) The recovery cannot produce in practice s_0 for an invalid *d*.

Proof. Indeed, any invalid *d*, say *d'* here, is by definition different from the valid one. Hence the

share it defines, namely $s = p|x'|d'$, is a noise share different of s_0. As we discussed already, chance of having then $H(s) = h$ are almost zero. Hence, in practice, no d' may ever lead to a successful match.

3) Guessing a discount code does not lower the complexity of any backup under $O(M)$

Proof. See [4].

4) A discount code d for backup B does not lower the complexity of any discounted recovery using d for a different backup B'. The latter remains $O(M)$ characterizing full-cost recovery of B'.

Proof. The discount codes being pseudo-random, it would be indeed like guessing in (2).

Property 3 means that the knowledge a discount code for a backup by the escrow, does not threaten any different backup at escrow's possession. A discount code once used by the escrow is thus of no further utility.

V. MULTI-SHARE NOISED SECRET SHARING

A. Rationale

The above discussed schemes used at the basis the 2-share secret sharing. Share s_0 was then noised, resulting in the, so-called, 2-share noised secret sharing. Complexity analysis sketched above and discussed in depth in [3], has shown that the 2-share noised secret sharing noised secret sharing schemes, for any given full-cost maximal recovery time R, provide the expected (full-cost) recovery time $E(R)$ equal to at most $R/2$. The static scheme provides exactly that expected value, while the scalable one may provide slightly less. These values are immediate and easy to spot consequences of the $O(M/2)$ average full-cost complexity for the static scheme.

One rationale of these properties is a uniform distribution of the suffix r (or r') of the noised share within I or I' for $m > 0$. One consequence is that for any probability p, the recovery time may be over $(1-p)R$. For instance for $p = 10\%$, it would be at least $0.9R$, i.e., almost twice as big as the user could expect most likely. By the same token, it could be also under $0.1R$. The cloud costs would be in accordance. Some users may be expected to feel uneasy with such a relatively likely perspective of the almost double bill. In turn, intruders may feel attracted to gamble over the uniformly likely perspective of the cheaper that on the average disclosure. In both cases, there is room for schemes where R is closer at will to $E(R)$. As it will appear, this is the property of the $(k+1)$ – *share noised secret sharing* schemes we introduce now. These schemes generalize the schemes above, assimilated to $k = 1$, towards larger k values $k = 2,3,4\ldots$ We keep the notation from Section 2 and after, adding eventually obvious indices.

B. Backup Creation

To start, O chooses k and, as before, the prohibitive 1-node recovery time D. The rationale for "best" choice of k will appear soon. Next, O defines a (usual) $(k+1)$ – share secret with thus the random shares $s_0 \ldots s_{k-1}$ and $s_k = K \text{ XOR } s_0 \text{ XOR } s_1 \ldots \text{ XOR } s_{k-1}$. Each s_j ; $j = 0 \ldots k-1$; has the same structure as s_0 above, namely $s_j = p_j | r_j$. Every suffix r_j is also as before adjusted to be under the *size of the noise space* M, if the need occurs. We provide soon the way to define M that slightly differs from the previous one. O computes then k hints $h_0 \ldots h_{k-1}$, where $h_j = H(s_j)$ for every j. A *match attempt* for any noise x, to be performed during the recovery, will consist of the calculation $H(f_j + x) ?= h_j$ for specific j.

It may occur that all the noises and hints are explored for a successful full-cost recovery. To choose then M conform to D, O starts with the measure or an estimate of the *throughput* T that is the number of 1-node match attempts per time unit, basically a second. Then M is chosen as $M = D\,T / k$. Indeed, for every M, there are $k\,M$ match attempts possible. Hence $D = k\,M / T$. Note that for $k = 1$ the M choice is compatible with that above for the 2-share noised secret sharing. Next, O forms P that is now $P = (p_0 \ldots p_{k-1}, h_0 \ldots h_{k-1}, M)$. Finally, O sends out the couple (s_k, P) as the K backup. O retains also the vector $d = (d_0 \ldots d_{k-1})$ as the discount code, where every d_j spans (the same) number m of the suffix bits of its s_j.

C. Recovery

To recover K, the legitimate user sends out the request $P_d = (P, R, d)$. Escrow sends then to the cloud all the backup data except for s_k. The match attempts split over the N cloud nodes, as before, Fig. 4, using the static or the scalable scheme. For the latter scheme, the coordinator defines N as $N = k\,M' / T\,R$. The rationale is first that for every noise x' within the reduced noise space, i.e., $x' = 0, \ldots, M' - 1$, the node in charge of attempting the matches for some x embedding x', i.e., for some d_j within d we have $x = x'|d_j$, has in fact k such match attempts to generate. These are all the attempts $H(f_j + x) = H(f_j + x'|d_j) ?= h_j$ for every $j = 0 \ldots k-1$. Notice that up to now we had $k = 1$ only, hence a single match attempt for every x'. Next, to meet the R bound, the node has to perform at most $R\,T$ attempts, assuming that all the nodes provide the same throughput (a homogeneous cloud). On the other hand, with the already discussed hash partitioning, the node performs in practice $k\,M' / N$ attempts. Hence, we have $k\,M' / N \leq R\,T$. The coordinator of the static scheme chooses the minimal possible N that is the one above. The scalable scheme generates N in a more involved way. The cloud is supposed heterogeneous, we recall, hence T may vary among nodes. The calculus is then distributed among the nodes as we mentioned already.

The N finally generated is usually somehow larger than the size-optimal one, [2].

Regardless of the partitioning scheme, every node attempts the matches for each y assigned to it as already discussed. Every node determining x for some h_j, i.e., encountering a successful match, reports the share $s_j = f_j + x$ to the coordinator. Once the coordinator gets every expected s_j, it reports them to the escrow. That one performs the XORing of all of them with s_k and sends the recovered secret, i.e. K, to the user. As for $k = 1$ above, the full-cost recovery must normally (without any cloud infrastructure failure) succeed for every k. A discounted one accordingly always succeeds in practice for a valid discount code and always fails otherwise.

D. Discussion

The recovery computation has obviously the maximal complexity $O(M')$, for every k. The average one is now $O(M' \; k \; / \; (k + 1))$. Accordingly, the average recovery calculation time is $R * k / (k + 1)$. The value $k / (k + 1)$ is known as the average value of the largest randomly chosen value in interval [0, 1[, among k such choices. Obviously it increases towards 1 with k. For the 2-share noised secret sharing we match the already discussed values $O(M' / 2)$ and $R / 2$. For higher k, both raise up towards M' and R' respectively. For instance, if O chooses a 6-share secret sharing, hence k =5, and the recovery requestor U expects a 5-minute recovery, U may choose $R = 6$ min. Instead of choosing $R = 10$ min for a 2-share only scheme. A perhaps up to 40% lower cloud bill in consequence. Also, the probability of the recovery time being within some fraction yR is now y^k instead of only y for a 2-share only scheme. Hence, in our example, it is 10^{-5} instead of 10^{-1} only. One may expect thus to deter much more strongly any gambling temptations of an insider on the escrow site.

A possible inconvenient of a larger k may be a k times longer d. The key owner wishing to only memorize it, as one does it for the credit card, may have trouble for $k > 3$. A way out may be parts of a one-way hash of a *code phrase*. That one should be, for the owner, easy to remember or reconstruct. It should also be harder to crack by known tools than a full-cost recovery itself. This can result, as usual for passwords, from mix of letters, numbers and special characters. It could be as easy nevertheless as, e.g., "In Jan. 1950 my age was 3.5 years". For, say, 1B discount code per share and our $k = 5$, any five bytes of the hash could do. The generation of the k pseudo-random shares should be amended consequently. There are various easy ways to do it.

VI. RELATED WORK

The basis for the work above is the static and the scalable schemes in [2]. These schemes use both hash partitioning. In [3], one proposes also an RE_{NS} scheme using the range partitioning. Our discounted recovery calculation may be expected applying to this scheme as well. Another scheme in [3], noises multiple shares, i.e., it provides the $(k + 1)$-share noised secret sharing with $k > 1$, like we do in Section 5. However, whether that scheme may be generalized to support discounts is at present an open question.

Besides, we are not aware of any other related work specific to some kind of discounted recovery. The work related to the noised secret schemes in general, including the overview of various proposals for key recovery, is extensively discussed in [2] and in [3]. We thus avoid repeating it here. We only notice however that a bird's eye view may assimilate a discount code to a particularly easy to use trapdoor decryption function d_m, of bit-length m. The "power" of successive functions for a key: d_1, d_2… scales up then exponentially with m.

VII. CONCLUSION

Discounts appear a potentially highly useful capability for an RE_{NS} scheme. The existing 2-share schemes generalize towards discount management easily. The key becomes "never-lost", with (illegal) disclosure cumbersome at will for the attacker and yet with cheap (legal) and practical recovery by the discount possessor. This combination, unique up to now for a key recovery scheme, has the potential to offset the current fears of key loss. On the one hand, our schemes may help users managing sensitive data purely locally, but fearing key loss or data disclosure anyhow. On the other hand, they may aid those attracted by big data outsourcing. Who remained deterred up to now, by fears of accordingly big consequences of key loss, or, by perhaps equally calamitous consequences of cloud content disclosure.

With respect to the work in progress, we continue the analysis of our schemes. We further plan to extend it to other schemes in [3].

[1] ARO Meeting on Cloud Security. GMU March 11-12,2013. http://csis.gmu.edu/albanese/events/march-2013-cloud-security-meeting/ .

[2] Jajodia, S., Litwin, W., Schwarz, Th., S.J. Recoverable Encryption through a Noised Secret over a Large Cloud. 5th Inl. Conf. on Data Management in Cloud, Grid and P2P Systems (Globe 2012). Springer Verlag, Lecture Notes in Comp. Sc.

[3] Jajodia, S., Litwin, W., Schwarz, Th., S.J. Recoverable Encryption through a Noised Secret over a Large Cloud. Intl. Journal on Large-Scale Data and Knowledge-Centered Systems, TLDKS IX, LNCS 7980, 2013.

[4] Jajodia, S., Litwin, W., Schwarz, Th., S.J. Key Recovery Using Noised Secret Sharing with Discounts and Large Clouds. Lamsade Research Report. July 2013 http://www.lamsade.dauphine.fr/~litwin/cours98/CoursBD/Key%20Recovery%20with%20Discounts%20Res%20Rep.pdf

[5] Raluca Ada Popa, Redfield, C., Zeldovich, N. & Balakrishnan, H. CryptDB: Protecting Confidentiality with Encrypted Query Processing. SOSP '11, October 23–26, 2011, Cascais, Portugal.

[6] Smith, K. How Practical Is Computable Encryption? In[1].