Reviews and author rebuttal from ACM CCS 2021; followed by changes in this submission

Review #147A
===========================================================================

Overall merit
-------------
4. Weak Accept

Reviewer expertise
------------------
2. Some familiarity

Paper summary
-------------
This paper provides a formal security treatment of the HLL algorithm in the adversarial setting. Their consider the status quo impl where the hash function is un-keyed along with a keyed version. The authors show that the former is vulnerable to attack while the latter resist them for the most part.

Strengths
---------
A formal and well-done security analysis of HLL. Given that this algo is reportedly widely used, this work tackles an important problem. The analysis is thoughtfully done and comprehensive, given the intended scope. It also lays the groundwork for analyzing future versions of the algo. The editorial quality is also very good.

Weaknesses
----------
The core contribution is just a security analysis of an existing scheme. It is also unclear exactly how big of an impact the attacks outlined in the paper would have on real-world products. The paper quality is quite high but I'm just a bit concerned with the scope/ impact of it.

Comments for author
-------------------
The presentation could be simpler if you define h : {0,1}* -> Z_{2^m} * Z_{2\ell}. Having to parse bit strings is an unnecessary distraction.

Questions for authors? response
---------------------------------
could you make the impact of your attacks a bit more concrete?

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Review #147B
===========================================================================

Overall merit
-------------
4. Weak Accept

Reviewer expertise
------------------
3. Knowledgeable

Paper summary
-------------
Hyperloglog (HLL) is a streaming algorithm that allows a user with small memory to observe a very large stream of data elements and to estimate the number of distinct elements seen in the stream.
In it's essence, the algorithm inserts each observed element $x_i$ into a deterministic random function H and then keeps track of the maximum number of 0s seen in any of the hashes H(x_i), which can be done space efficiently.
If the number of 0s is k, then with high-probability there were around 2^k distinct elements in the stream.
The concrete HLL algorithm contains some more details like averaging to reduce variance and an alternate approach for small cardinalities.

The focus of this paper is to study this streaming algorithm in an adversarial setting.
The adversary's goal is to design a stream of elements that results in the maximally incorrect estimate, i.e. real stream contains many distinct elements, but estimator says that it only contains a few.
Several different adversaries with varying degrees of knowledge/power are studied and the authors present successful attacks on HLL.
Lastly, the authors show how to 'immunize' HLL against the presented attacks

Strengths
---------
The paper studies a very interesting question with the potential of real-world impact, since HLL is used in various real deployed systems.
The authors try to cover a wide variety of possible attack scenarios and also attempt to related their attacks to concrete systems such as Redis.
The paper is well written and easy to follow for the most parts.

Weaknesses
----------
The paper currently only provides a somewhat superficial study of the applications of their attacks and the actual real-world impact is somewhat unclear to me.
All attacker models assume that the adversary has access to the

"random" hash functions, which is not well enough motivated in my opinion. These seem to be algorithm internal parameters that are chosen randomly and I am not fully convinced why an external adversary should know them.

Section 8 seems to be a bit of a mess and I had a hard time following the details. It would be great if the authors could clean up this section and provide a more rigorous write-up.
The actual content of this section should be really simple though, if I understand correctly. The solution for immunizing a HLL is basically to take away the adversary's access to the random hash functions, which are now modeled as a keyed-PRF. This seems equivalent to modeling the random hash functions as "cryptographically secure" and to assume that the adversary has no access to them.

Comments for author
-------------------
I would really appreciate, if the reviewers could focus on one system like Redis and provide a more concrete study of it. For instance, what is the concrete adversarial model, i.e. does it know the hash functions or can it recover them. How is this adversarial model justified in a concrete real-world scenario. What would be the attack from start to finish in that concrete application.

Questions for authors? response
---------------------------------
Please provide a concrete attack example from start to finish that justifies your attacker model.
How is it reasonable to assume that the adversary has access to the random functions, but not access to the keyed-PRF in your solution.

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

Review #147C
===========================================================================

Overall merit
-------------
2. Weak reject

Reviewer expertise
------------------
4. Expert

Paper summary
-------------
This paper studies the resilience of the hyper LogLog (HLL) sketch (used to estimate the cardinality of a multi-set) to an adversary trying to attack its integrity.††The authors consider four different

classes of attackers that vary in how much access they have to the sketch, from only having black-box access to a replica sketch to having full access to the registers used by the sketch.

They show, extending on attacks from prior work, that if the hash function used is public, the adversary can always get the sketch to (significantly) undercount the number of unique items in the sketch.††The proposed attacks mainly work by identifying items that can be added without increasing the count stored in the sketch, either by hitting buckets that have already been hit, or by ensuring that the number of leading 1s is small.

The authors show that these simple attacks can lead to significant undercounts.††

Additionally, the authors consider what happens when a private (i.e., keyed) hash function is used in place of a public hash function.

They show that as long as an adversary cannot distinguish H(x) from random (i.e, H is a PRF) the Hyper Log Log sketch preserves accuracy.

They propose a new simulation-style security definition and prove integrity of the resulting sketch.

Thus, the main result seems to be that if the adversary knows H, then he can find items that can be inserted into HLL without increasing its count, thus resulting in an undercount.††The attacks show how to use this with varying levels of access to the sketch.

Strengths
---------
- The LogLog sketch and its variants such as Hyper LogLog are widely used in practice and thus it is important to understand the integrity guarantees they provide against malicious adversaries.

- The paper is well written and easy to follow

Weaknesses
----------
- The attacks are specific to a particular variant of the HLL sketch.††Specifically, they don?t really attack the sketch (i.e., the data structure of the underlying Flajolet-Martin sketch) but instead take advantage of the particular algorithm used to estimate cardinality given the sketch. Thus, it is not clear how general the result is.

- The proposed defense (as the authors admit) destroys the mergeability of the resulting sketch. This seems like a critical limitations as in many (most?) settings mergeability is a crucial property when multiple sketches are built and then combined. I would have liked to see at least some discussion on ways to maintain mergeability with a private hash.

- The comparison to prior work is insufficient. Specifically, the authors state that they improve on the attacks from refs [8] and [10]. I would have appreciated a brief description of these attacks and why (and how much) their attacks improve upon these.††The authors state that [8] and [10] may have been evaluated on different versions of HLL, and while this may indeed be true, it seems reasonable for the authors to reimplement the prior attacks to see how much improvement their attacks get. A table summarizing these results would improve the paper.

Comments for author
-------------------
Overall, I felt that this was an interesting paper, but the focus on particular implementations of HLL made the scope a bit too narrow for CCS. In particular, I would have liked to see an analysis of how well your attacks may generalize to other variants of LL, and also more considerations of defenses.

Additional comments:
p.4:
- You introduce four adversary models S1-S4.††You explain that these come from prior work, but don?t provide motivation for all of them.††In particular, why is S1 a realistic model?††Where would you have a shadow HLL that is exactly identical to the real one?††Moreover, one that you can reset.††Even if using a public hash function, if that hash function is salted, it seems unreasonable to assume that the shadow version would have the same salt.††Moreover, if S1 is not identical (i.e., uses a PRF with a different random key), then it seems like it would not help break security.††So, please justify this modeling.

- Additionally, you don?t really seem to analyze S3 analyzing S2 and S4 instead.††So, why introduce S3?††

p.6:
- the description of the complex attack is not complete.††In particular, you do not describe the reprocessing of the list L

p.7:
- How does your sketched attack in S1 compare to the one from [8]?††There is not enough description or analysis here to be able to assess whether your proposal is an improvement.

p.10:
- You should stress in your proof of security (and discussion) that the adversary is not given oracle access to F_k(\cdot).††Note that this is allowed by the standard definition of a PRF, but (I think) would destroy the security of your construction.

Rebuttal Response
====================================================================

We thank the reviewers for the detailed comments and look forward to interactive discussions in the days ahead. In this initial response, we are forced to focus on key points raised by the reviewers.

Review #147A

We agree that using a bit-level description is distracting, but we wanted to stick as closely as possible to the original presentation of HLL, for the benefit of readers.

On concrete impact: we presented the attacks abstractly, since they are completely generic (see discussion bottom right of p.1). However, the introduction describes several scenarios where HLL is used and where cardinality manipulation would have an impact. See also the discussion in Section 7.1 which breaks [5], a prominent NSDI paper using HLL to detect network attacks. Since submission, we have implemented attacks on Redis described in Section 6. They perform exactly as expected. So if one used Redis? HLL ?off the shelf? as part of the system in [5], then a DoS attacker could evade detection.

Review #147B

The reviewer states that the adversary knows algorithm internal parameters that are chosen randomly and wonders about the practical relevance of this assumption. The reviewer?s statement is incorrect.

While the original analysis of HLL [1] assumes that the hash function behaves like a random oracle, the HLL implementations available online use fixed parameters and hash function that can be read off from source code. See for example:

https://github.com/redis/redis/blob/unstable/src/hyperloglog.c#L396 ;
https://github.com/redis/redis/blob/unstable/src/hyperloglog.c#L466

and

https://github.com/prestodb/presto/blob/2ad67dcf000be86ebc5ff7732bbb9994c8e324a8/presto-main/src/main/java/com/facebook/presto/type/khyperloglog/KHyperLogLog.java#L20

for Redis and Presto.

If one switches to using a keyed PRF, then of course that key has to be protected. If (for example) a keyed version of HLL is implemented on a router and used to monitor network traffic, the attacker can still try to manipulate the traffic; security would then rely on the adversary not being able to extract the key from the router.

Regarding the rigorousness of Section 8: Section 8.2 provides intuition, while Section 8.3 is rigorous. If the referee could point to something specific that they find non-rigorous in Section 8.3, we'd be glad to address it. The reviewer is correct in their intuition for the security proof (except that there is only one fixed hash function, which we replace by a PRF). As we wrote in the introduction: "... it is possible to provably secure HLL against cardinality manipulation by the simple expedient of replacing its internal hash function h by a keyed, variable-input-length pseudo-random function (VIL-PRF)".

Regarding the request for a concrete study of Redis: Section 6 gives our analysis and attacks for Redis. Since submission we have implemented our attacks against Redis. See the response to Reviewer A.

Review #147C

The attacks are not specific to a particular variant of the HLL sketch: we study both ?classic HLL? and the Redis implementation of HLL (which does indeed use a different cardinality estimator).

Mergeability is trivial when using the same key PRF K in all sketches, similarly to TLS load balancing. We consider mergeability with multiple keys a challenging open problem, see Section 9.

On comparison to related work: we compare our results to [8] in Section 7. We obtain much higher reduction factors (e.g. almost 1000 instead of the 5 reported in [8]), implying that our attack is much better. [10] studies the privacy properties of HLL, and is not concerned with cardinality estimation. So no direct comparison to [10] is possible. We only reuse their adversarial models since we found them to capture interesting sets of adversarial capabilities.

On models: We state that we do not consider S1 to be very realistic but are forced to analyse it since it is used in [8]. We introduce S3 because this is a practical adversarial setting. We don't have attacks in S3 that are better than those in S2. See the last paragraph of Section 4.3 where this is explained.

On list reprocessing in the complex attack: this is identical to the simple attack, and is described above both attacks on p.6. We can clarify this.

Of course the adversary is not given oracle access to the F_k(\cdot)!! If he was, we would be back to the original attack scenarios, and no security would be achievable. And, in general, we don't give an adversary direct access to underlying primitives used in cryptographic constructions when they are keyed. We have concrete examples in mind and can discuss them in detail with the reviewers.

Comment @A1 by Reviewer C
---------------------------------------------------------------------------
We thank the authors for their rebuttal.

After much internal discussion, the reviewers felt that while this paper certainly has merit, it is a bit too narrowly focused for CCS

at this point. While pointing out that simple attacks work against
real-world systems, as is done here, is important, we felt that the
paper needed more discussions of generalizations of the attacks or
protections against this class of attacks. We encourage the authors
to continue down this exciting line of research and hope they find
an appropriate venue for this work.

## Changes in this Euro S&P 2022 submission

We have included links to the relevant parts of the Redis and Presto
source code in footnote 5, to try to make it clear that specific,
important implementations of HLL use fixed parameters and hash
functions (as in the S2, S3 and S4 settings). This is in an attempt
to avoid any reviewer confusion concerning fixed vs random hash
functions, cf. ACM CCS 2021 Reviewer B.

We have included experimental results for the Redis implementation
of HLL in Section 7.2 and Table 4 (this implementation was already
analysed theoretically in the previous version of the paper). This
addresses a request from ACM CCS 2021 Reviewer B.

We have reworked Section 8.3 (formal security analysis) to try to
make it easier for readers to understand. This responds to a comment
from ACM CCS 2021 Reviewer B who described it as "a bit of a mess"
but did not expand on this comment despite the request to do so in
our rebuttal.