9-2017

Genomic security (Lest we forget)

Tatiana BRADLEY
University of California, Irvine

Xuhua DING
Singapore Management University, xhding@smu.edu.sg

Gene TSUDIK
University of California, Irvine

DOI: https://doi.org/10.1109/MSP.2017.3681055

Published in IEEE Security and Privacy, 2017
September, Volume 15, Issue 5, Pages 38-46
http://doi.org/10.1109/MSP.2017.3681055

Genomic Security (Lest We Forget)

Tatiana Bradley | University of California, Irvine
Xuhua Ding | Singapore Management University
Gene Tsudik | University of California, Irvine

Genomic privacy has attracted much attention from the research community, because its risks are unique and breaches can lead to terrifying leakage of sensitive information. The less-explored topic of genomic security must address threats of digitized genomes being altered, which can have dire consequences in medical or legal settings.

As full genome sequencing becomes increasingly practical and affordable, it's not hard to imagine a (near) future where large numbers of people store and maintain their digitized genomes. Ubiquitous access to one's digitized genome opens the door to a wide range of applications, ranging from serious (for instance, disease screening or paternity testing) to social (for instance, ancestry tracing or compatibility/dating). At the same time, a genome represents a veritable gold mine of extremely personal and sensitive information about its owner as well as that person's ancestors, descendants, and siblings. Furthermore, as the ultimate static biometric, a leaked genome can't be revoked or modified, thus exacerbating privacy concerns. Consequently, genomic privacy is a very timely and important subject, which has, in recent years, understandably attracted much attention from the research community. (See "Whole Genome Sequencing: Revolutionary Medicine or Privacy Nightmare?" for an overview of genomic privacy challenges.[1])

With the spotlight on the privacy front, where moderate progress has been made, comparatively less attention has been devoted to genomic *security*. This is surprising because security is at least as important as privacy. In the context of personalized medicine, a modified genome can lead to wrong drugs or treatments being prescribed or administered. In terms of paternity or common ancestry testing, a modified genome can yield incorrect test results, which can translate into equally incorrect legal decisions.

Some recent work on genomic security (for instance, G.K. Ragesh and K. Baskaran's "Cryptographically Enforced Data Access Control in Personal Health Record Systems"[2]) focused on access control for health records, which—though important—doesn't prevent the possibility of an insider modifying genomic data. In particular, Ragesh and Baskaran sought to prevent, rather than detect, unauthorized modifications.[2]

One possible reason for genomic security *not* having received much attention thus far is that it's perceived not to pose any new challenges. In this article, we show that this conventional wisdom might be unjustified. After taking a closer look at genomic security, we identify some new challenges that can't be resolved by

naively applying current techniques. These challenges stem from several factors, including the size and longevity of the human genome, an unconventional application model, bandwidth and computation complexities, and the need to balance security with privacy.

Genomic Security

We envisage a generic application scenario with the following key features:

- An individual—Alice—obtains her digitized genome from an authorized sequencing lab (SL).
- Alice stores the result on her personal device, for instance, a laptop or smartphone.
- Later on, Alice wants (or is mandated) to conduct a genetic test, the purpose of which might be legal, medical, or social.
- The test requires Alice to provide some specific genomic data—typically, a small portion of the entire genomic sequence—to the application server (tester) that actually performs the test.

This scenario triggers various security issues for all stakeholders. One important issue is certification and periodic recertification of sequencing labs, because Alice clearly needs to trust the SL to correctly sequence and digitize her genome. This process would likely be done by a trusted government agency, for instance, the US Food and Drug Administration (FDA).

Another issue is certification of application-specific servers, which could be trickier due to a wide range of medical, legal, and social applications, each with its own access requirements to specific genomic excerpts. This would let Alice decide which parts of her genome should be revealed to a particular application. For example, a social app might be restricted to accessing segments that determine certain physical characteristics, such as height and hair or eye color, whereas a legal DNA profiling app that uses the short tandem repeat (STR) method might be restricted to accessing Combined DNA Index System (CODIS)-stipulated 13 core loci. This diversity calls for a well-defined policy or authorization syntax such that an application server can be certified to permit access to only a set of fixed and specific genomic locations or ranges thereof.

A related issue is proving rightful ownership; for example, if the test is conducted remotely (that is, over the Internet), how does Alice convince the tester that she supplied her own genomic data? This clearly requires a certification scheme that involves all three entities—the individual, the lab, and the tester.

Despite the obvious importance of all of the above, we focus in this article on more basic issues: the authenticity and integrity of Alice's genomic data in the context of diverse applications. At first glance, this seems easily addressable via textbook security techniques, such as hash functions and digital signatures. However, as we discuss, the problem is a bit more challenging than it appears.

Genome Representation

In general, the human genome is a sequence of 3.2×10^9 base pairs—two letters chosen from the tiny four-letter alphabet: {adenine (A), cytosine (C), guanine (G), and thymine (T)}. The simplest way to represent it digitally is to use an array of three-bit blocks, each representing the first letter of a base pair at the corresponding absolute position. (An additional bit might be needed to account for sequencing errors, for example, a symbol "X" where a base letter was unreadable). The second letter doesn't need binary representation as it can be deduced from the first one using the base-pairing rule (www.biology-pages.info/B/BasePairing.html).

However, because human genomes have a high degree of similarity, an individual's genome is often represented as a set of differences with respect to a fixed reference genome. In practice, only approximately 3×10^6 base pairs are needed for most genetic applications. Hence, although a full and complete representation of a single genome might take up to 200 Gbytes, a compact version based on a reference representation, for instance, using the 1,000 Genomes Project variant call format (www.internationalgenome.org/wiki/Analysis/vcf4.0), occupies only about 120 Mbytes. For simplicity's sake, we assume the genome reference representation is a list of 3×10^6 tuples of the form: (x, L_x), where L_x is the base pair at position x. In practice, L_x might contain more complex genomic data regarding position x. Nonetheless, the value of L_x doesn't impact the security issues discussed later.

Stakeholders and Trust Model

Again, the stakeholders in the aforementioned scenario include the individual—Alice, the SL, and the application server—tester. For now, we assume that the SL operates mostly offline, whereas Alice and the tester interact over the Internet or another similarly insecure communication channel. Given proper and timely certification by a higher authority (for instance, the FDA), we assume that everyone trusts the SL. However, the tester doesn't trust Alice regarding the authenticity and integrity of her genomic data. At the same time, Alice doesn't trust the tester with any of her genomic information beyond that which the latter is authorized to access for the particular test.

In the future, the SL's role might be replaced by a personal sequencing device. Such devices, though certainly not affordable today, are already available from vendors

such as Illumina. In the extreme, we can imagine a world in which individuals own and operate their own sequencing devices, perhaps as part of or as an attachment to a smartphone. Naturally, it would be crucial for such a device to be equivalent to an SL in terms of both functionality and trust. In particular, it would have to be certified by a trusted authority and would need to incorporate secure hardware coupled with some degree of tamper resistance as well as a means of secure logging and auditing.

Requirements

The first requirement is an efficient means for Alice to convince the tester of her genomic data's integrity and authenticity.

The second requirement is privacy of Alice's genome: because a typical genomic test uses only a small portion of the entire genome, the rest must be kept secret from the tester. Ideally, information revealed by Alice mustn't allow the tester to learn anything else about Alice's genome. However, this is unrealistic from the outset because a genome isn't random; information that corresponds to certain loci might allow the tester to infer (with absolute certainty, or at least with nonnegligible advantage over a random guess) contents of other loci. Although privacy is a key goal, the inference problem is beyond the scope of this article.

The third requirement is performance: minimal storage, communication, and computation overheads incurred by all stakeholders. This is of highest importance for Alice who might be using a resource-constrained personal device. Of course, following current trends, Alice could outsource storage and computation of her genomic data to a cloud service provider (CSP), which has vastly greater resources than her device. There's still an incentive to minimize all costs, due to the CSP's very large scale of both storage and computing. Outsourcing neither changes the trust model above nor invalidates the requirements. The other two stakeholders—the SL and the tester—are expected to be commercial entities with ample computing, storage, and communication facilities. (An exception would be peer-to-peer social genomic applications, in which a tester might be another personal device.) Nonetheless, it's always desirable to reduce their overheads.

Challenge

A prominent challenge stems from the conflict between security and privacy requirements. On one hand, Alice's privacy implies that she should control her genomic information revealed to the tester. On the other hand, the tester demands authenticity and integrity, which means that Alice must be unable to modify (or delete parts of) her digitized genome.

This issue is exacerbated by the compact reference representation. Consider a simple example. Suppose that the tester requests a sequence of X base letters, starting at position Y. We assume that Alice's genome has just one difference in this range: an A at position Y' (for $Y' - Y < X$). The next difference is a C at position $Y_{\text{nxt}} \geq Y + x$, while the previous difference is a G at position $Y_{\text{prv}} < Y$. An honest Alice would send the tester a single tuple: (Y', A). She would also attain maximal privacy by revealing nothing beyond the minimum required by the tester.

Alternatively, a malicious Alice could cheat and send an empty string, thus claiming that her genome and the reference have no differences in the range $[Y,Y + X]$. If we assume that each difference is somehow individually authenticatable (for example, signed by the SL at sequencing time), Alice can't create base letter differences where none exist. However, she can easily omit actual differences from the requested range. In the database security literature, this is sometimes called the *range query completeness* problem, where a more generic term "records" is used instead of "differences." It also has a trivial solution: adjacent differences must be securely (cryptographically) bound, that is, authenticating a difference at position Y' must allow the tester to securely determine that previous and next differences occur at positions Y_{prv} and Y_{nxt}, respectively.

This method is readily applicable in our context; for example, for each difference at position Y' involving a base letter $L_{Y'}$, SL could sign a tuple: $(Y', L_{Y'}, Y_{\text{prv}}, Y_{\text{nxt}})$ where Y_{prv} and Y_{nxt} are as defined before, with two special symbols (for instance, –inf and +inf) indicating the start and end. For each difference in the requested range, Alice would send the tester one such signed tuple, and any cheating on her part would be trivially detectable. If Alice really had no differences in the entire $[Y,Y + X]$ range, there would necessarily exist either (or both) the closest previous or next closest difference, represented as a distinct signed tuple. It's easy to see that if Alice provides an SL-signed tuple corresponding to either position, the tester can verify it and thereby determine that Alice's genome has no differences in the $[Y,Y + X]$ range.

Although secure, this approach sacrifices some of Alice's genomic privacy. Note that, in the above example, the tester learns (potentially a lot) more than it's entitled to learn. Specifically, regardless of the number of differences in the $[Y,Y + X]$ range, the tester learns the positions of two other differences: Y_{prv} and Y_{nxt}. There seems to be no easy solution to this.

As this discussion illustrates, reconciling privacy and security isn't obvious, at least if reference representation is used. In the rest of this article, we discuss ways to

simultaneously attain integrity, authenticity, and completeness for the tester as well as privacy for Alice.

Naive Approaches

We start with some very naive approaches to authenticity and integrity. Though not quite practical, they provide insights into ensuing design challenges and lead us to a somewhat practical baseline technique.

No Privacy

In the no privacy (NoP) approach, after sequencing, the SL signs the compact genome representation and references its owner's identity (Alice) and/or the owner's public-key certificate. Thereafter, Alice can easily prove authenticity and integrity to the tester by transferring the whole signed genome and authenticating herself in the process. This incurs for Alice the lowest possible costs for storage (just the cleartext genome) and computation (almost none). In return, Alice has no privacy whatsoever, while communication overhead is maximal. The tester's costs are similar to Alice's, albeit storage is needed only temporarily, up to signature verification.

Finer-Grained Privacy

In the finer-grained privacy (FGP) approach, the SL partitions Alice's genome sequence into segments and separately signs each, using some unique identifier to tie all the segments together as well as to bind them to Alice. This way, Alice sends the tester the smallest set of signed segments that contain necessary/requested positions and base letters. One possibility is to pick uniform-size segments, which makes for easier processing and storage. Alternatively, genomic specialists can determine segments of variable lengths according to the application needs; for example, standard test types might call for specific fixed ranges. We don't pursue this further as it's orthogonal to our study. This approach offers weak privacy for Alice because it leaks extra (not strictly required) information to the tester. The actual amount of leakage depends on the segmentation algorithm and the specific tester application.

Baseline: Extreme FGP

Taking FGP to the extreme, we can obtain an optimal mix of security and privacy at the expense of storage. In this case, called extreme FGP (eFGP), the SL uses the full genome representation, instead of the compact (reference-based) version—that is, it individually signs every single base letter along with its position. As a result, Alice attains optimal privacy because only data corresponding to requested (and, presumably, duly authorized) positions is revealed. For its part, the tester can individually authenticate each position/base-letter pair and verify ownership.

eFGP's tradeoff is in performance: all parties incur much higher costs than NoP. SL has to compute 3.2×10^9 signatures. With RSA, the minimum near-term safe key/modulus size is 2,048 bits (anticipated to be secure until 2030), while elliptic curve cryptography (ECC) needs 224 bits for roughly the same security. (Both 2,048-bit RSA and 224-bit ECC are believed to offer 112 bits of security.) We can discount the SL's computation complexity because, as a commercial entity, it has ample resources and can always find a way to pass the extra costs onto its customers. Alice doesn't need to verify individual base-letter signatures; at delivery time, the SL can supersign the whole genome separately, and Alice can verify just that one signature.

Of more concern is storage, that is, space complexity: even if we ignore storage for position metadata, signatures themselves result in data expansion of two to three orders of magnitude, depending on the signature type. This translates into hundreds of gigabytes (ECC) or nearly a terabyte (RSA) per genome. For Alice, storing this much data on a personal device, and communicating it, is likely to be prohibitive in the near future. On the other hand, assuming that a typical test involves only 0.1 percent of the genome (which approximates the typical difference between any two humans), Alice's communication with the tester would be commensurately less intensive, that is, 1,000 times less.

For the tester, eFGP requires as many signature verifications as the number of base letters requested from Alice. This is where the choice of the signature scheme matters most. For instance, it's well known that, with small public exponents, RSA is generally 10 to 30 times faster than elliptic curve (EC) digital signature algorithm (DSA) for verification. The next question is whether the extra bandwidth consumed by RSA signatures is outweighed by faster verification. The answer depends on several variables, such as network speed and requested plaintext size.

Consider the following example. On a commodity 2015 MacBook Pro, OpenSSL reports signature verification speeds of 15,702/s and 1,540/s for RSA and ECC, respectively. We assume a 1-Gbps network and equally capable interfaces for Alice and the tester. Also, the tester can pipeline signature verification, that is, verify each base-letter signature immediately on receipt. We set $k = 3.2 \times 10^6$, which corresponds to 0.1 percent of the genome, and RSA and ECC sizes of 2,048 and 224 bits, respectively. Then, RSA transfer delay is estimated as $(2{,}048 \times 3.2 \times 10^6)/10^9 \approx 6.5$ s, and DSA as $(224 \times 3.2 \times 10^6)/10^9 \approx 0.7$ s. These delays are clearly dwarfed by signature verification times: $3.2 \times 10^6/15{,}702 \approx 203.8$ s for RSA and $3.2 \times 10^6/1{,}540 \approx 2{,}078$ s for ECC. If we pick a much smaller $k = 1{,}000$, signature verifications would be 0.064 s for RSA and

0.65 s for ECC, while transfer delays remain relatively insignificant: 0.002 s for RSA and 0.0002 s for ECC.

Consequently, at least for the time being, RSA has a clear performance advantage. It's easy to see that the gap would grow significantly larger with bigger key sizes, for example, 3,072 and 256 bits. Although other tester CPUs could yield very different results, it seems unlikely (though not impossible) that ECC would outperform RSA, unless congestion or other factors drastically reduce network speed. Today, very low network speeds can be encountered if Alice and the tester communicate over a 2G or 3G cellular network; however, gigabit cellular is already available and will probably become pervasive in a few years.

Note that although virtually all modern signature algorithms use the well-known hash-and-sign technique, our earlier discussion ignores the cost of hashing, because it's assumed to be negligible compared to that of signature verification. In addition, all signatures in eFGP are computed on distinct plaintexts, because each "message" includes a base letter, its position, and a reference to Alice's identity (and/or her public-key certificate).

An auxiliary issue is storage (disk) read speed on Alice's device. Although disk read speeds of modern smartphones don't yet match top network speeds, commodity laptops easily reach gigabits/second disk read speeds, for example, MacBook Pro in 2015. We can safely assume that smartphones will catch up in a few years. Note that storage write speed on the tester's side is less important because of presumably abundant resources.

In summary, eFGP offers a useful baseline: it achieves the best balance between security for the tester and privacy for Alice. Its main drawback is performance.

Performance Optimizations

Here we consider some means of improving the baseline eFGP's performance.

Batch Verification

One natural way to speed up the tester's computation is by using batch signature verification. This way, Alice still sends the same data to the tester, which accumulates all plaintext hashes and all signatures and verifies the entire collection at the cost of one signature verification. (In other words, an accumulated hash is verified against an accumulated signature.) The best-known example is the batch version of full-domain hash (FDH)-RSA,[3,4] an RSA variant that requires an FDH—a cryptographic hash function that yields digests of the same bit size as the RSA modulus. However, batch FDH-RSA requires computing separate accumulators of message hashes and signatures, which costs $2k$ modular multiplications, where k is the number of signatures.

Because plain RSA signatures can be used safely with a fixed small public exponent of 3, each signature verification (without batching) entails two modular multiplications, resulting in the same $2k$ total. Therefore, there appears to be no performance gain for the tester in using batch FDH-RSA. In fact, the latter might be more expensive because FDH can be slower than a plain hash function.

Though batch techniques aren't unique to RSA, most others either require different public–private exponents per message or are applicable to batching signatures by multiple signers.

Condensed and Aggregated Signatures

Another potential optimization is condensed signatures,[5] which is very similar to batch verification, except that it's Alice who accumulates all signatures (by the same signer) into a single condensed signature and sends it, along with all plaintexts, to the tester. The latter accumulates all plaintext hashes and verifies one signature. Condensed signatures appear to be a perfect match for RSA because of its comparatively large signature size. Similar to batch, an FDH-RSA variant must be used here. Assuming a small public exponent, the tester computes only k (rather than $2k$ in batch RSA) modular multiplications, although Alice is now forced to compute the other k to produce the condensed signature.

There are also more general techniques, such as aggregated signatures, exemplified by the BGLS (Boneh, Gentry, Lynn, and Shacham) signature scheme.[6] BGLS and its follow-ons allow k signatures produced by k signers over k distinct messages to be aggregated into one signature. By verifying this signature against all k messages, each message's authenticity and integrity are ascertained. (As mentioned earlier, all base-letter messages are unique.) Also, BGLS doesn't require signers to be distinct; in fact, it's more efficient when all aggregated signatures are by the same signer. Aggregation performed by Alice requires k EC multiplications. As with condensed RSA, bandwidth overhead is minimal. The tester's verification requires k EC multiplications and one signature verification (pairing).

On one hand, k modular or EC multiplications performed by Alice is a costly endeavor, because her personal device might be computationally weak. On the other hand, Alice can precompute a condensed or aggregated signature. Furthermore, bandwidth savings can be substantial, for example, close to 6 s for $k = 3.2 \times 10^6$ in our example above. It thus remains unclear whether there's a performance incentive as far as using condensed signatures, unless bandwidth complexity must be minimized or precomputation by Alice is free.

Merkle Hash Tree

A popular tool in computer security, a Merkle hash tree (MHT) is a data structure for efficient authentication of any member or subset of a large set. It's a (typically, binary) tree where leaves are hashes of individual set members, and each interior node is the hash of its two children. Assuming a suitable cryptographic hash function, the tree root is the collective indirect hash of all leaves. The root node's signature thus authenticates the entire tree. In an MHT with n leaves, given an $O(\log n)$-size co-path, any set element (leaf) can be authenticated as being part of the tree by hashing upward toward the root and verifying the root signature. Constructing an MHT takes $O(2n)$ hashes and one signature. It's necessary to store only the leaves and the (signed) root because all interior nodes can be reconstructed with $O(n)$ hashes. One notable application for MHTs is efficient certificate revocation checking.

We can easily adopt the MHT construct to the problem at hand, as follows. The SL constructs Alice's MHT with the ordered sequence of base letters serving as the leaves, then signs the root. Alice reveals a genomic segment—a sequence of contiguous base letters—to the tester. To do so, Alice provides the segment and a co-path consisting of all sibling nodes on the path(s) from the root to the common ancestor(s) of the segment. The tester reconstructs the Merkle tree's root and verifies the SL's signature. Co-path length is bounded by the MHT height of approximately $32 \approx \log_2 3.2 \times 10^9$. Thus, Alice sends the tester up to 32 hashes (8 Kbits total at 256 bits/hash) and a root signature in addition to the requested base-letter segment.

For a k-long segment, this method involves negligible bandwidth overhead and only requires the tester to perform a single signature verification as well as $2k + 32 - \log k$ hashes.

One issue is Alice's storage: the entire tree takes more than 200 Gbytes with a 256-bit hash function. A well-known way to cut the storage cost by half is for Alice to reconstruct the tree at runtime. Then, Alice's storage would be the same as in NoP. However, the downside is the need to compute 3.2×10^9 hashes on demand, which is impractical.

Another issue is Alice's privacy: Alice reveals only what's absolutely necessary—that is, the requested segment base letters. Unfortunately, the co-path gives away additional information. Consider the example in Figure 1: leaves 2 through 6 correspond to the base-letter segment CGATA. The accompanying co-path would include nodes 1 and 12, but not base letters in positions 1, 7, and 8. However, knowledge of node 1 allows the tester to learn G, and node 12 can be used to learn T and G in positions 7 and 8, respectively. This is due to the low entropy of individual base letters; there are only four possibilities for 1, and 16 possibilities for 7 and 8, which makes exhaustive searching easy. Of course, a co-path node's height exponentially influences the complexity of an exhaustive search. Given an interior node at height z, $4^{(2^z - 1)/2}$ trials are necessary, on average, to learn its descendant leaf base letters. Therefore, an exhaustive search is practical up to about $z = 5$, implying that up to 32 extra base letters might be learned by the tester.

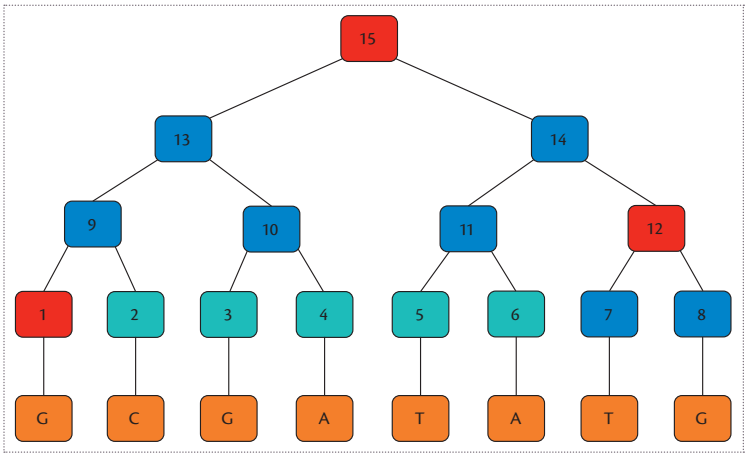

Figure 1. Merkle hash tree (MHT) leakage example. Knowledge of node 1 directly reveals leaf G, and knowledge of node 12 lets the tester know the leaves T and G for position 7 and 8, respectively.

Salted Merkle Tree

The natural next step is to prevent privacy leakage in MHT. This can be achieved using a salted Merkle hash tree (sMHT). For each base letter L_i at position i, the SL generates a pseudorandom salt s_i. The corresponding leaf LF_i is computed as $F_{s_i}(L_i, i)$, where $F()$ is a keyed pseudorandom function indexed on s_i, for instance, HMAC. An alternative is $LF_i = H(s_i, L_i, i)$ where $H()$ is a cryptographic hash function.

The rest of the tree is constructed as before. All salts are given to Alice by the SL as part of the initial digitized genome transfer. Salt bit size should be sufficient to rule out brute-force attacks, that is, at least 128. Then, Alice sends the tester all requested base letters along with their salts. This is in addition to the signed root and the co-path.

sMHT offers the same privacy for Alice, as well as the same integrity and authenticity guarantees for the tester, as eFGP. However, sending salts consumes additional bandwidth, comparable to eFGP without condensed or aggregated signatures. Unlike signatures, salts can't be compressed or accumulated. Salts also impose much higher storage overhead for Alice. There is a trivial way to avoid it if the SL generates all salts using a keyed pseudorandom function with a key K_a, for instance,

$s_i = F'_{K_a}(i)$, and shares K_a with Alice as part of the initial transfer. Then, Alice can easily recompute, on demand, all salts corresponding to the leaves in the revealed base-letter segment.

Another issue with both MHT and sMHT is the number of contiguous segments revealed to the tester. Our discussion above assumed only one such segment of variable size. It's quite possible that some genetic tests require many segments from disparate places in the genome. In that case, bandwidth and computational complexity of eFGP (with or without condensed/aggregated signatures) is unaffected, whereas tree-based techniques would require multiple partial (up to the height of the least common ancestor of all segment-formed subtrees) co-paths, one for each segment.

Redactable Signatures

An alternative approach for balancing authenticity and integrity for the tester with privacy for Alice is to replace standard signatures (for instance, RSA or EC-DSA) with specialized methods. One attractive concept is *redactable signatures* (RS), introduced more-or-less concurrently by both Robert Johnson and his colleagues and Ron Steinfeld and his colleagues.[7,8] An RS scheme allows authorized "cryptographic redactions" of a signed message. In other words, given a redactable signature with a signed message, an authorized party can redact the message and obtain a new valid signature, without knowledge of the signer's signing key. In the context of RS, we can view a base-letter segment as a "redaction" of the full genome, in which all other data is crossed out. SL computes a redactable signature over Alice's genome. This signature is then redacted to suit the specific segment to be sent to the tester.

Because RS is a very general concept, both eFGP and sMHT can be viewed as redactable signature schemes; indeed, very similar approaches are described by Johnson and his colleagues.[7] (We note that they also suggest salted MHTs.[7] However, because they're computed in a special way, salts for revealed base letters don't need to be transmitted, as they can be recomputed by the tester.) Several RS variations have been proposed, for example, hiding sizes of redacted areas[9] and RS over nonstring data.[10] However, these features appear irrelevant to the context of genomic security.

Signature Aggregation and Chaining

The final approach we discuss is digital signature aggregation and chaining (DSAC).[11,12] It's a very simple technique, similar to the one sketched out earlier for secure range queries. It provides authenticity, integrity, and completeness. The basic idea is to construct signatures over a sequence of elements such that it becomes easy to demonstrate authenticity, integrity, and completeness of a reply to any range query. Given a genomic sequence $\{L_1, \ldots, L_N\}$, the SL computes a signature chain in two steps, for $0 < i \leq N$:

- $R_0 = s_0, R_i = [L_i, i, s_i, H(R_{i-1}, s_{i-1})]$ and
- $\sigma_i = F_{sig}(R_i)$,

where F_{sig} is any suitable hash-and-sign signature function, $\{s_0, \ldots, s_N\}$ are $N + 1$ pseudorandom salts (same as in sMHT), and $H()$ is a hash function. Without getting into further details, it's easy to see that to authenticate and verify integrity and completeness of a reply to a range query $[i,j]$, it suffices to produce $H(R_{i-1}, s_{i-1})$ as well as $\{L_i, \ldots, L_j\}, \{s_i, \ldots, s_j\}$ and σ_j.

From the bandwidth perspective, this is a particularly appealing technique due to its minimal overhead. However, DSAC's most attractive aspect is the verification cost: $(j - i)$ hashes with salts and one signature validation of σ_j. The downside of DSAC is its storage cost, which is as large as eFGP.

Limitations of Current Techniques

We gave an overview of several fairly simple approaches to genomic security. All offer roughly equivalent security (authenticity and integrity) for the tester. As far as Alice's privacy, eFGP, sMHT, and DSAC offer the best privacy by revealing only the required information. As far as performance, eFGP with condensed/aggregated signatures has the lowest possible bandwidth overhead, although computation overhead amounts to $O(k)$ multiplications for Alice and the tester. sMHT has very low computation overhead dominated by $O(2k + 32)$ hashes and a signature verification, while its bandwidth overhead is slightly higher, unless many disparate (noncontiguous) segments are involved. Finally, DSAC also offers very low bandwidth overhead coupled with the only k hashes and one signature verification.

To compare performance, Table 1 estimates several overhead factors, including the number of signatures the SL computes, the number of signatures the tester verifies, the number of bits Alice stores and transmits, and the number of cryptographic operations Alice performs.

As Table 1 shows, although all schemes except NoP offer optimal security and privacy, none incurs overheads close to the lower bounds. For example, in the case of sMHT, Alice stores approximately 214,000 times and transfers approximately 28 times more data; this is in addition to the 32-fold computation cost.

Improving Efficiency

Further work is needed to reduce computation overhead. One obvious step is to avoid the full genome

Table 1. Performance comparison of a realistic sample set of variable values: $N = 3.2 \times 10^9$; $N_r = 3.2 \times 10^6$; $k = 1{,}000$; $s_\sigma = 2{,}048$; $s_h = 256$; and $s_s = 128$.*

Approach	Sequencing lab (no. of signatures computed)	Tester (no. of signatures verified)	Alice's workload		
			Storage (bits)	Communication (bits)	Computation (no. of hash operations)
No privacy	1	1	$3N_r + s_\sigma$	$3N_r + s_\sigma$	–
Extreme finer-grained privacy (eFGP)	N	k	$3N + s_\sigma N$	$3k + s_\sigma k$	–
eFPG + aggregation	N	1	$3N + s_\sigma N$	$3k + s_\sigma$	$O(k)$
Merkle hash tree (MHT)	1	1	$3N + 2s_h N + s_\sigma$	$3k + s_h \log N + s_\sigma$	$O(\log N)$
Salted MHT	1	1	$3N + 2s_h N + s_s N + s_\sigma$	$3k + s_s k + s_h \log N + s_\sigma$	$O(\log N)$
Digital signature aggregation and chaining	N	1	$3N + s_\sigma N + s_s N$	$3k + s_s k + s_\sigma$	–
Lower bound	**1**	**1**	$\mathbf{3N_r + s_\sigma}$	$\mathbf{3k + s_\sigma}$	**–**

N is number of base pairs in full genome representation; N_r is number of base pairs in reference representation; k is number of base pairs requested by the tester; s_σ is signature bit size; s_s is bit size of salt; and s_h is bit size of hash function digest.

representation, which takes a heavy storage toll. Ideally, the SL would sign a reference representation of Alice's genome and grant Alice the ability to redact arbitrary portions of this representation, which are outside the range requested by the tester, as well as efficiently prove that nonredacted portions (all properly signed by the SL) are complete—that is, Alice hasn't omitted anything from the requested range.

We sketch out one possible approach that satisfies these requirements and offers an optimal tradeoff among security, privacy, and efficiency. The main idea is for the SL to sign all pairs of adjacent mutations, similar to the trivial method we described earlier. However, actual positions and contents of mutations aren't revealed; instead, the SL signs cryptographic commitments to both contents and positions of adjacent mutations. Each signed tuple contains two commitments. A reference representation with k mutations would need $k + 1$ signed tuples. Note that two dummy sentinel mutations are needed to demarcate the beginning and end of the genome. When the tester requests all mutations in a specific range, Alice supplies one or more tuples. If the positions of both mutations in a tuple are within range, Alice decommits their locations and contents. (The tester can easily verify correctness.) If the lower-indexed mutation is within range and the higher one isn't, Alice decommits only the former. She then proves (in zero knowledge) that the other mutation's committed value (position) is greater than the upper range limit. A similar process is followed if a signed tuple's higher-indexed mutation is in the range while the lower one isn't. In the case in which the requested range contains no mutations, Alice releases a single signed tuple, wherein the lower-indexed mutation is below the lower range limit, and the higher-indexed mutation is above the upper range limit. She then provides two zero-knowledge proofs, each showing that committed positions are outside the requested range. Proving that a committed (and secret) integer is within a specific range is both possible and quite efficient, using techniques such as those offered by Fabrice Boudot.[13]

Due to length restrictions for the present article, we don't elaborate on this approach.

Anonymity

In the context of some genetic (for instance, parentage) tests, Alice might want to hide her identity from the tester. For *pseudonymity*, it suffices for the SL to tie Alice's genome to a random pseudonym or a pseudonymous public-key certificate. Alice can then communicate with the tester over some anonymous channel, such as Tor. Stronger privacy (that is, anonymity) requires that any two genetic tests must be unlinkable. Clearly, none of the methods described above is unlinkable. However, there is some hope for redactable signatures, which can be made unlinkable, as shown in "Composable

and Modular Anonymous Credentials: Definitions and Practical Constructions."[14]

We argue that genomic security has been underappreciated in favor of privacy. We believe security is vital to adoption of emerging and future personal genomic applications. The interesting mix of integrity, authenticity, and privacy requirements for multiple parties translates into a research challenge. We explored several fairly intuitive approaches, none of which satisfies all ideal security and performance requirements. Clearly, much remains to be done. ■

References

1. E. Ayday et al., "Whole Genome Sequencing: Revolutionary Medicine or Privacy Nightmare?" *IEEE Computer*, vol. 48, no. 2, 2015, pp. 58–66.
2. G.K. Ragesh and K. Baskaran, "Cryptographically Enforced Data Access Control in Personal Health Record Systems," *Procedia Technology*, vol. 25, 2016, pp. 473–480.
3. M. Bellare, J.A. Garay, and T. Rabin, "Fast Batch Verification for Modular Exponentiation and Digital Signatures," *Proc. Int'l Conf. Theory and Applications of Cryptographic Techniques* (EUROCRYPT 98), LNCS 1403, Springer, 1998, pp. 236–250.
4. M. Bellare and P. Rogaway, "The Exact Security of Digital Signatures—How to Sign with RSA and Rabin," *Proc. Int'l Conf. Theory and Applications of Cryptographic Techniques* (EUROCRYPT 96), LNCS 1070, Springer, 1996, pp. 399–416.
5. E. Mykletun, M. Narasimha, and G. Tsudik, "Signature Bouquets: Immutability for Aggregated/Condensed Signatures," *Proc. European Symp. Research in Computer Security* (ESORICS 04), LNCS 3193, Springer, 2004, pp. 160–176.
6. D. Boneh et al., "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," *Proc. Int'l Conf. Theory and Applications of Cryptographic Techniques* (EUROCRYPT 03), LNCS 2656, Springer, 2003, pp. 416–432.
7. R. Johnson et al., "Homomorphic Signature Schemes," *Topics in Cryptology*, LNCS 2271, Springer, 2002, pp. 244–262.
8. R. Steinfeld, L. Bull, and Y. Zheng, "Content Extraction Signatures," *Proc. Int'l Conf. Information Security and Cryptology* (ICISC 01), LNCS 2288, Springer, 2001, pp. 285–304.
9. E.-C. Chang, C.L. Lim, and J. Xu, "Short Redactable Signatures Using Random Trees," *Topics in Cryptology*, LNCS 5473, Springer, 2009, pp. 133–147.
10. C. Brzuska et al., "Redactable Signatures for Tree-Structured Data: Definitions and Constructions," *Proc. Int'l Conf. Applied Cryptography and Network Security* (ACNS 10), LNCS 6123, Springer, 2010, pp. 87–104.
11. M. Narasimha and G. Tsudik, "DSAC: Integrity for Outsourced Databases with Signature Aggregation and Chaining," *Proc. ACM Int'l Conf. Information and Knowledge Management* (CIKM 05), 2005, pp. 235–236.
12. M. Narasimha and G. Tsudik, "Authentication of Outsourced Databases Using Signature Aggregation and Chaining," *Proc. Int'l Conf. Database Systems for Advanced Applications* (DASFAA 06), 2006, pp. 420–436.
13. F. Boudot, "Efficient Proofs That a Committed Number Lies in an Interval," *Advances in Cryptology* (EUROCRYPT 00), Springer, 2000, pp. 431–444.
14. J. Camenisch et al., "Composable and Modular Anonymous Credentials: Definitions and Practical Constructions," *Advances in Cryptology* (ASIACRYPT 15), LNCS 9453, Springer, 2015, pp. 262–288.

Tatiana Bradley is a PhD student in computer science at the University of California, Irvine (UCI). Her research focuses on applied cryptography, including privacy-preserving computation. Contact her at tebradle@uci.edu.

Xuhua Ding is an associate professor at the School of Information Systems of the Singapore Management University. His research interests include trustworthy computing, system security, applied cryptography, and multimedia security. Ding received a PhD in computer science from the University of Southern California (USC). Contact him at xhding@smu.edu.sg.

Gene Tsudik is a Chancellor's Professor of Computer Science and director of the Secure Computing and Networking Center at UCI. His research interests include topics in security and applied cryptography. Tsudik received a PhD in computer science from USC. He's a Fellow of ACM, IEEE, and AAAS and a foreign member of Academia Europaea. Contact him at gts@ics.uci.edu.