Supplementary Materials for

Deriving genomic diagnoses without revealing patient genomes

Karthik A. Jagadeesh,* David J. Wu,* Johannes A. Birgmeier, Dan Boneh,† Gill Bejerano†

*These authors contributed equally to this work.
†Corresponding author. Email: dabo@cs.stanford.edu (D.B.); bejerano@stanford.edu (G.B.)

Published 18 August 2017, *Science* **347**, 692 (2017)
DOI: 10.1126/science.aam9710

This PDF file includes:

Materials and Methods

Patient datasets

Whole exome sequences of patients were obtained from dbGaP studies phs000204.v1.p1 (*6*) (Freeman-Sheldon Syndrome), phs000244.v1.p1 (*7*) (Miller Syndrome), phs000295.v1.p1 (*8*) (Kabuki Syndrome), and phs000477.v1.p1 (Hajdu-Cheney Syndrome). Pre-processed variant call format (VCF) files for patients from 2 Centers for Mendelian Genomics were obtained from dbGaP studies phs000693.v4.p1 (University of Washington), and phs000711.v3.p1 (Baylor Hopkins). Our trio family was obtained from Stanford Hospital. All human subject research was performed under guidelines approved by the Stanford Institutional Review Board.

Sequencing reads were mapped to the GRCh37/hg19 assembly of the human genome using BWA MEM v0.7.10-r789 (*31*). Variants were called using GATK v3.4-46-gbc02625 following the HaplotypeCaller workflow from the GATK Best Practices (*32*).

Variant annotation

ANNOVAR v527 was used to annotate variants with predicted effect on protein coding genes using gene isoforms from the ENSEMBL gene set version 75 for the hg19/GRCh37 assembly of the human genome (*33*, *34*). All canonical gene isoforms were used where the transcript start and end are marked as complete and the coding span is a multiple of three.

Cryptographic techniques

In a secure multiparty computation (MPC) protocol (*35*, *36*), a group of users (often called *parties*) seek to jointly compute a function over their inputs without revealing any additional information about their particular inputs. The function that the parties compute is determined based on the specific scenario. The computation consists of several rounds of interaction, where in each round, the parties exchange a series of messages. At the conclusion of the protocol, each participant learns the output of the computation evaluated on everyone's joint input. No additional information beyond the explicit output is revealed to any party (Fig. S2).

Every arithmetic computation can be expressed as a sequence of Boolean logical operations (that is, operations on bits {0, 1}). This is precisely how the modern computer works. Yao's protocol allows two users, Alice and Bob, to compute arbitrary functions over their inputs. More precisely, if Alice has an *input x* and Bob has an input *y*, Yao's protocol allows them to *compute f(x,y)* in a way such that Alice learns nothing about y and Bob learns nothing about *x* other than the output value *f(x,y)*. In general, expressing a function in terms of Boolean operations greatly increases the computational cost of evaluating the function. To maximize the efficiency of Yao's protocol, it is important to choose functionalities with simple or compact representations as Boolean circuits (Fig. S4).

In this work, we cast diagnosis of Mendelian patients as (simple) arithmetic/logic computations that admit efficient Boolean circuit representations. We now describe how the secure computation protocols work. To do this we first introduce two standard tools from cryptography: symmetric (secret-key) encryption (*37*) and oblivious transfer (*38*–*40*).

Encryption and decryption

A secret-key encryption scheme consists of two functions: Encrypt and Decrypt (Fig. S5). The encryption function takes a cryptographic key k and a message m and outputs a ciphertext c (Fig. S5B). The decryption function takes the cryptographic key k and a ciphertext c and outputs a message m (Fig. S5C). Intuitively, encryption and decryption are inverse operations: if we encrypt a message under a key k, decrypting the resulting ciphertext with the *same* key k recovers the original message. More precisely, we can say that for any key k and any message m, Decrypt(k, Encrypt(k, m)) = m. In a *symmetric* (or secret-key) encryption scheme, both the encryption and the decryption functions require knowledge of the secret cryptographic key. The key is a random string drawn from some key-space. The precise nature of the key-space varies depending on the details of the encryption scheme, and is immaterial to our presentation in this paper. An encryption scheme is considered to be secure if the ciphertext does not reveal any information about the underlying message to any user who does not possess the secret encryption key (certainly, a user who holds the secret key can decrypt and learn the message). One way to formalize this is to say that a user who does not have the encryption key is unable to tell an encryption of a message m_0 apart from an encryption of another message m_1. In other words, ciphertexts hide all information about their underlying message to all users who do not have the encryption key (Fig. S5D).

Under this definition, messages can also be encrypted multiple times. For instance, a message m can be "double encrypted" under two keys k_1 and k_2 by first encrypting m using k_1 and then encrypting the resulting ciphertext using the second key k_2. This procedure yields another ciphertext. Decryption proceeds by first decrypting with key k_2, and then decrypting the result (a ciphertext) with k_1. In particular, we can write

$$\text{Decrypt}(k_1, \text{Decrypt}(k_2, \text{Encrypt}(k_2, \text{Encrypt}(k_1, m)))) = m$$

Security of the double encryption scheme follows directly from the security of the underlying encryption scheme. In particular, a user who does not have both k_1 and k_2 cannot learn any information about the underlying message that has been doubly encrypted using k_1 and k_2. Numerous symmetric (secret-key) encryption schemes exist in the literature (*37*).

Oblivious transfer

An oblivious transfer (OT) protocol (*38–40*) is a two-party protocol between a sender and a receiver. An OT protocol enables the receiver to selectively obtain one of two possible messages from the sender without revealing to the sender which message the receiver requested. More precisely, the sender holds two messages, denoted k_0 and k_1 and the receiver holds a selection bit $b \in \{0,1\}$. At the end of the OT protocol, the receiver obtains the chosen message k_b and learns nothing about the other message k_{1-b}. The sender does not learn anything about the receiver's choice bit b. Numerous oblivious transfer protocols have been proposed in the literature (*38–40*).

Overview of steps for secure computation

In a secure two-party computation protocol, Alice holds an input $x \in \{0,1\}^n$ and Bob holds an input $y \in \{0,1\}^n$. We write $\{0,1\}^n$ to denote a binary input of length n (e.g., for instance, n could be of the binary representation of the variant vector or the gene vector we define in our main text). Their goal is to compute a function $f(x, y)$ on their

joint input *(x, y)*. The computation is considered "secure" if at the end of the computation, the only information that Alice and Bob learn is the function value *f(x, y)* and nothing else about the other party's input. It is important to note here that the function output *f(x, y)* could reveal some information about the inputs *x* and *y* (for example, in our trio scenario, whatever de novo variant we report in the child, we can deduce by definition does not exist in either parent). In this paper, we work in the semi-honest (honest-but-curious) model where we assume that Alice and Bob follow the protocol specification as directed, but may, at the end of the protocol execution, try to infer some additional information about each other's private input. We now describe how Yao's protocol (Fig. S2) can be used to securely evaluate any function over two inputs in the semi-honest model.

To apply Yao's protocol, it is first necessary to represent the function *f* as a Boolean circuit on inputs *x* and *y*. At the most basic level, the building blocks we have are AND (*x* AND *y* = True only if *x* = *y* = True, otherwise it is False) and XOR (exclusive or, *x* XOR *y* = True only if *x* = True and *y* = False, or if *x* = False and *y* = True, and otherwise it is False) gates. These basic gates can be combined to obtain circuits of arbitrary expressive functionalities (*41*). In our description below, we will oftentimes refer to the concrete example of securely evaluating the AND function on single-bit inputs (above).

Overview of Yao's secure two-party computation protocol

We now describe Yao's protocol. For ease of presentation, we present a simplified (but less efficient) description of Yao's protocol here. Our implementation (based on the JustGarble (*42*) library) follows the high-level blueprint described here, but includes several optimizations, notably the free-XOR (*43*) and half-gate (*44*) optimizations.

Step 1 (Fig. S2B): For each wire in the circuit, Alice chooses two keys. Recall that in a Boolean circuit, each wire in the circuit can take on two possible values (0 or 1; sometimes also referred to as False and True, respectively). Alice associates one of the keys with the wire value 0 and another key with the wire 1. For the particular case of securely evaluating the AND function $z = x \wedge y$, Alice picks three pairs of keys and associates one pair with each of x, y, and z. We denote these keys $k_x^0, k_x^1, k_y^0, k_y^1, k_z^0, k_z^1$. In this example, k_x^0 is the key associated with the input bit x taking on the value 0 and k_z^1 is the key associated with the output bit z taking on the value 1.

Step 2 (Fig. S2C): For each gate in the circuit, Alice constructs a "garbled" truth table. For each row in the truth table, the algorithm takes the key associated with the value of the output wire and *double* encrypts it using the two keys associated with the values of the two input wires. For the particular case of evaluating a single AND gate, Alice would construct the following table of ciphertexts

- $e_1 = \text{Encrypt}\left(k_x^0, \text{Encrypt}\left(k_y^0, k_z^0\right)\right)$
- $e_2 = \text{Encrypt}\left(k_x^0, \text{Encrypt}\left(k_y^1, k_z^0\right)\right)$
- $e_3 = \text{Encrypt}\left(k_x^1, \text{Encrypt}\left(k_y^0, k_z^0\right)\right)$
- $e_4 = \text{Encrypt}\left(k_x^1, \text{Encrypt}\left(k_y^1, k_z^1\right)\right)$

For the output wires of the circuit, instead of double encrypting an encryption key, Alice directly double encrypts the value of the output wire (e.g., 0 or 1).

Step 3 (Fig. S2D): After garbling the circuit (Steps 1-2), the secure computation begins with Bob using the oblivious transfer protocol (above) to obtain the keys for the input wires associated with his input. The oblivious transfer protocol ensures the following: Bob only learns one of the two keys associated with each of his input wires (this will ensure that Bob can only evaluate the function on a single set of inputs), and Alice does not learn which wire Bob requested (that is, Alice does not learn Bob's input). For the particular case of evaluating a single AND gate, if Bob's input is $b \in \{0,1\}$, then Bob would play the role of the receiver in an OT protocol with input b. Alice would play the role of the sender with messages k_y^0, k_y^1 (the keys associated with Bob's input wire). At the end of the oblivious transfer protocol, Bob obtains k_y^b (the key associated with his input), and learns nothing about the key associated with the complement of his input (k_y^{1-b}). Alice learns nothing about Bob's input b.

Step 4 (Fig. S2E): After Bob receives the keys associated with his input via the oblivious transfer protocol, Alice sends Bob the garbled tables associated with each gate (after randomly permuting the rows of each table). Additionally, Alice sends Bob the wire encodings of her input. For the particular case of evaluating a single AND gate, if Alice's input is $a \in \{0,1\}$, Alice would send k_x^a (the key associated with $x = a$) to Bob.

Step 5 (Fig. S2F): With all this information, Bob can complete the function evaluation and compute the output. In particular, after Steps 3 and 4, Bob should have a single key for each of the input wires of the Boolean circuit. Then, for each gate in the circuit, Bob takes the input keys he has and attempts to decrypt the rows in the garbled table associated with that gate. Because the entries in the garbled table are *double encrypted* using the keys associated with the input wires to the gate and Bob only has a *single* key for each of the wires, Bob is only able to decrypt a *single* row in the garbled table.

Step 6 (Fig. S2G): In doing so, Bob is able to learn one of the keys associated with the gate's output wire (moreover, by construction of the garbled table, the output key Bob obtains is precisely the one associated with the value corresponding to evaluation of the gate on the input bits). Thus, starting with the input wires, Bob is able to evaluate the circuit gate-by-gate. Once Bob reaches the output layer of the circuit, he is able to decrypt the ciphertexts and obtain the value of each output wire.

Summary. To summarize, in Yao's secure two-party computation protocol, Alice begins by constructing a garbled truth table for each gate in the Boolean circuit. She does so by double encrypting each row in the truth table (using the keys associated with the input bits). She gives Bob the garbled truth tables as well as the keys associated with her input. Using oblivious transfer, Bob obtains the keys associated with his input. Armed with a single key for each of the input wires in the circuit, Bob is able to evaluate the garbled circuit gate-by-gate. For each gate, Bob takes his input keys and uses them to decrypt one of the rows of the garbled table associated with the gate. This yields the key associated with the particular wire. Finally, at the end of the computation, Bob decrypts the ciphertexts associated with the output wires to learn the output of the circuit. Bob then sends the result of the computation to Alice.

Extending Yao's protocol to *N* parties

Yao's protocol allows two parties to securely evaluate an arbitrary function. However, in general, we desire to compute across a large number of parties (e.g., study

participants). While there are secure multiparty computation protocols that support more than two parties, (e.g., the SPDZ (*45*), GMW (*36*), or BGW (*46*) protocols), a key limitation of these protocols is that they require all participating parties to be online during the protocol execution. Moreover, the number of rounds of communication in the protocol often grows with the complexity of the computation (note that this is in direct contrast with Yao's protocol which is a two-round protocol, regardless of how complicated the computation is). As a result, there are substantial engineering hurdles to deploying these general protocols for multiparty computation across a large number of parties. In some cases (e.g., BGW (*46*) and GMW (*36*)), the total bandwidth also scales quadratically in the number of parties, further limiting the practicality of these protocols.

A more efficient solution for general multiparty computation that avoids both the requirement that participating parties be online during the protocol execution as well as the potential communication blowup is to work in a "two-cloud" model. In this model, we assume there are two *non-colluding* cloud servers that facilitate the protocol execution. At the beginning of the protocol execution, each of the participating parties "split" their inputs and share it with the two cloud servers. As long as the two clouds do not collude with each other, they do not learn anything about the inputs to the computation. After the two cloud servers have received the inputs from each of the participating parties, they engage in a two-party secure computation protocol (such as Yao's protocol) to compute the function of interest. Notably, the parties that contributed the data do not have to be online during this step of the protocol. Moreover, communication is only necessary between the parties and the cloud servers; parties in particular do not have to communicate with each other. In a practical deployment, these two cloud servers might be managed by distinct governmental organizations within the NIH or WHO. Thus, by working in the two-cloud model, it is possible to transform any computation between n individuals into a secure two-party computation between two non-colluding parties.

Step 7 (Fig. S2H): We now describe how to secure evaluate any functionality in the two-cloud model. Suppose there are n parties participating in the protocol execution and let $x_1, \dots, x_n$ denote their private inputs. To securely compute a function f, each of the n participants chooses a random value r_i and sends r_i to one of the two cloud servers. They then send to the other cloud server the value $x_i - r_i$ (note that the subtraction is performed modulo a large integer N). Once every party has submitted their inputs r_i and $x_i - r_i$ to the two cloud servers, the first cloud server has a vector of random values $a = (r_1, \dots, r_n)$ and the second cloud server has a vector of random differences $b = (x_1 - r_1, \dots, x_n - r_n)$. Because the subtraction is taking place modulo N, the values in b are distributed uniformly and more importantly, independently of the x_i's. The pair $(r_i, x_i - r_i)$ is often referred to as an "additive secret sharing" of the input x_i. The property that this additive secret sharing scheme satisfies is that a single share reveals no information about the input, but two shares completely define the input. This means that as long as the two cloud servers do not collude, they learn no information about each party's input (since they each possess just one share of the secret).

To complete the secure computation (of a function f), the two cloud servers simply apply Yao's protocol to the following two-party functionality:

$$g\big((r_1, \dots, r_n), (x_1 - r_1, \dots, x_n - r_n)\big) = f(r_1 + x_1 - r_1, \dots, r_n + x_n - r_n) = f(x_1, \dots, x_n).$$

In other words, the two clouds compute the functionality that takes as input two vectors (each containing n values) and outputs the function f evaluated on the components corresponding to the sum of the two input vectors. Since summing the input vectors in this case reconstructs each party's input, this procedure corresponds precisely to evaluating f on the parties' inputs. Moreover, the two cloud servers do not learn any additional information about any particular party's input because the evaluation of g is performed using Yao's protocol (which is a secure two-party computation protocol). We emphasize that the method described here is general and can be used to evaluate any function on *arbitrary* input types. To secret share an arbitrary data item (which may not be numeric), we would take the binary representation of the data item and *interpret* it as the bits of a number modulo a large integer N. We can then apply the procedure described here to split the data item into two additive shares. Secure computation on the secret-shared values then proceeds exactly as before.

Constructing our Boolean circuits

As described above, arbitrary Boolean circuits can be constructed using only AND and XOR gates. To efficiently represent our set-intersection-based algorithms as Boolean circuits, we first construct some intermediate building blocks from the basic AND and XOR gates. The intermediate building blocks we require include addition circuits, comparison circuits, and equality circuits (Fig. S6). For these building blocks, we use the circuits by Kolesnikov, Sadeghi, and Schneider (*47*).

Software implementation

In our implementation, we use the JustGarble library (*42*) for our implementation of Yao's garbled circuits, and we use the Asharov-Lindell-Schneider-Zohner (*48*) implementation for the oblivious transfer protocols. For better performance, we also implement the half-gates optimization (*44*) for Yao's garbled circuits. For our benchmarks, we set up a client and server on Amazon EC2 (to simulate the two cloud providers), and measure the total compute time, bandwidth, and overall protocol execution time (taking into account the network communication). We run our experiments on two memory-optimized EC2 instances (M4.2xlarge). Each instance runs an 8-core 2.4 GHz Intel Xeon E5-2676 v3 (Haswell) processor and has 32 GB of memory. While our protocols are naturally parallelizable, we use a single thread of execution in all of our experiments, and do not take advantage of the available parallelism. To simulate the non-colluding two-cloud model, we used a wide-area network (WAN) setting where the two servers are far apart. We placed one of the servers on the West Coast (specifically, in the Northern California availability zone) and the other on the East Coast (specifically, in the Northern Virginia availability zone). The code we used for our experiments is available at https://github.com/dwu4/genome-privacy.

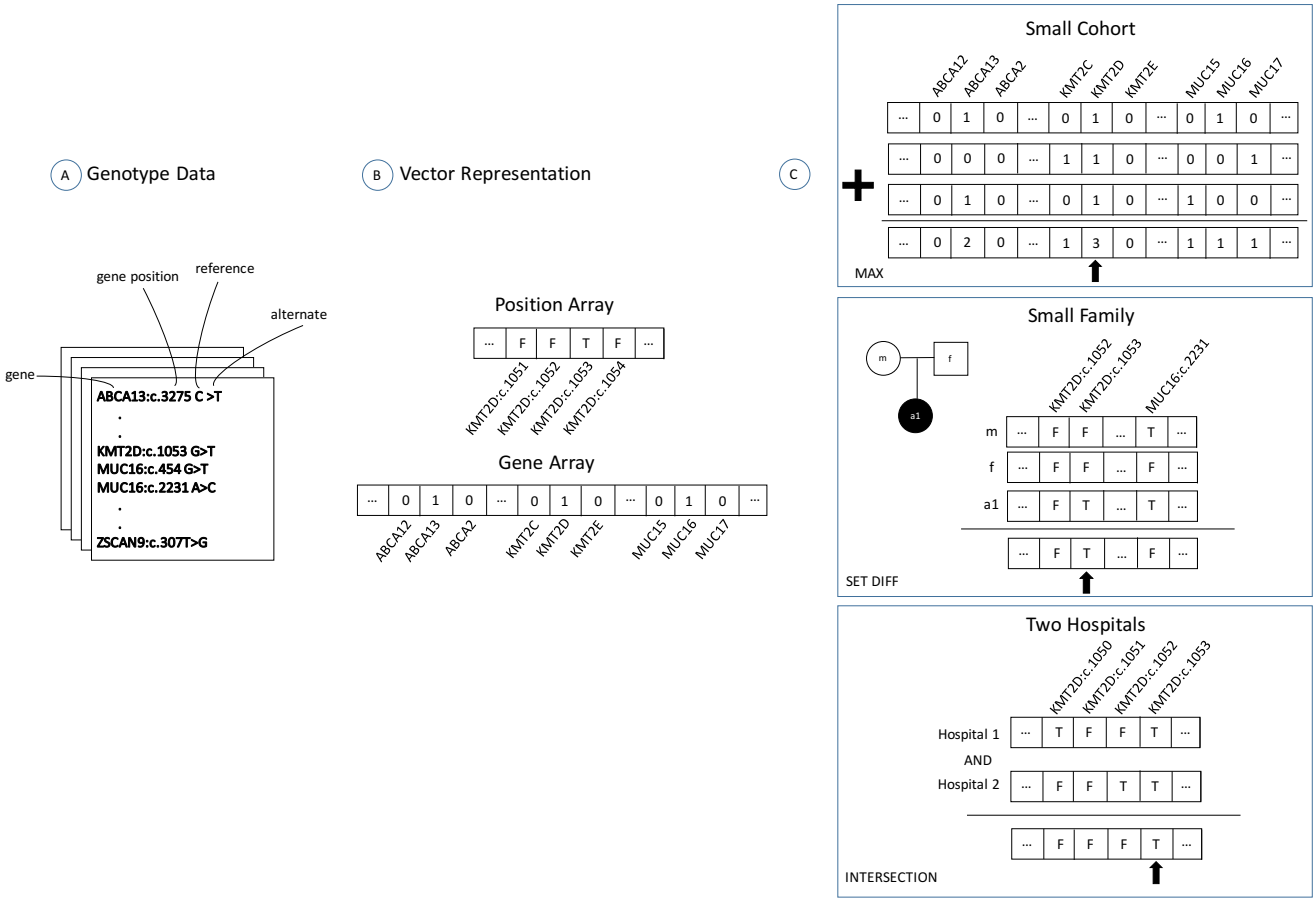


Fig. S1.

Representing genomic data as vectors for secure computation. **(A)** Each individual holds their personal genome private. **(B)** They are asked to fill in a position array/vector with True and False values depending on whether they have a rare functional variant at the listed position, or a 0/1 value in a gene array depending on whether they have none/some rare functional variant/s in each listed gene. A simple computer program can be used to streamline this step. **(C)** The resulting position/gene vectors are used to obtain the results of Table 1.

A

Alice holds a secret value a:
a = True or a = False.

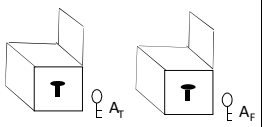
Bob holds a secret value b:
b = True or b= False.

Alice and Bob want to compute together
f(a,b) = a AND b
without Bob discovering anything about a,
or Alice discovering anything about b.

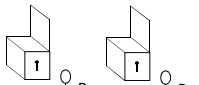
B

Alice prepares:

2 types of big boxes + keys
(for each value she may be holding)



2 types of small boxes + keys
(for each value Bob may be holding)



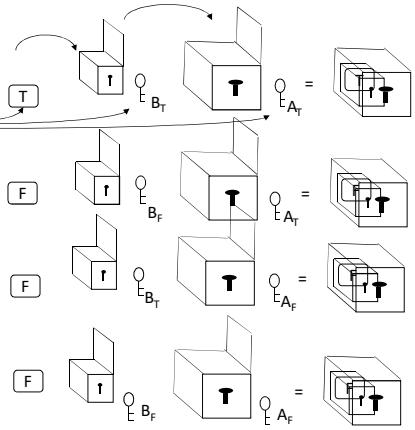
2 types of notes
(with the 2 possible outcomes)

The notes fit in the small boxes,
that lock and fit into the big boxes,
that also lock.

C

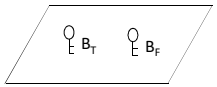
Alice prepares four big locked boxes
matching all four possible computations:

Alice's a	Bob's b	f(a,b) = a AND b	
T	T	T	T
T	F	F	F
F	T	F	F
F	F	F	F



D

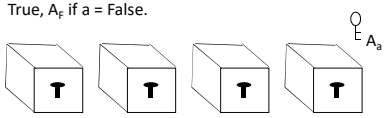
Alice puts on the table the two keys (labeled B_T and B_F)
she prepared for the small boxes and leaves the room:



E

<u>Implemented using an "Oblivious Transfer" Protocol</u>

Bob enters the room and picks up the appropriate key
"B_b": B_T if his secret value b = True, B_F if b = False. After
picking up his key, he leaves the room. Alice then gives
Bob all four unmarked big locked boxes. She also gives
him one unmarked key "A_a": A_T if her secret value a =
True, A_F if a = False.



F

Bob now holds 4 unmarked big locked boxes,
A key from Alice A_a, and his own key B_b.

He tries to get the note from all four boxes,
using A_a on the big boxes, and B_b on the small ones.

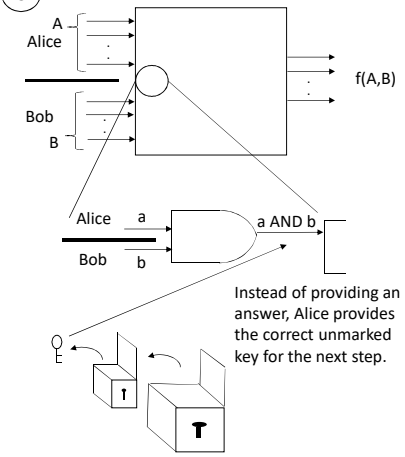
By design Bob can only reach a single note.

This note holds the correct answer for a AND b,
that Alice and Bob set out to compute together.

Alice has learned nothing about Bob's value b
(she has left the room before Bob picked his key).

Bob has learned nothing about Alice's value a
(he received from Alice an unlabeled key).

G



H

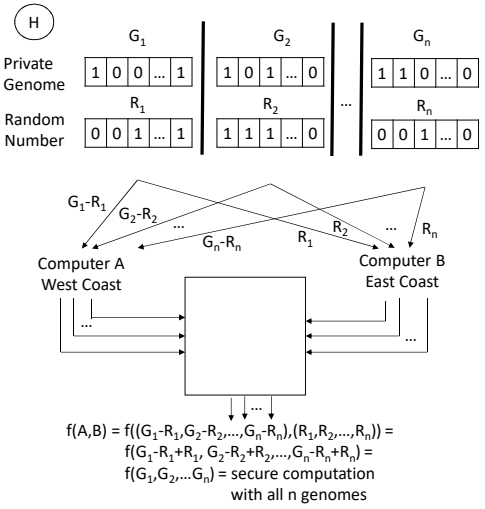


Fig. S2

Yao's protocol for secure multiparty computation. Steps 0-7 describe Yao's protocol for computing any function *F* between two or more parties *F(A, B, ... ,Y, Z)*. We first describe a secure two-party computation protocol between Alice (A) and Bob (B). **(A) Step 0:** Alice and Bob are trying to compute a joint function without revealing their inputs to the other party. **(B) Step 1:** Alice creates a key/box for each possible value for each input (0 or 1). **(C) Step 2:** Alice double locks (double encrypts) each of the four possible outputs by placing the relevant output note in two boxes corresponding to each combination of the two inputs. **(D) Step 3:** Alice gives Bob the option of choosing exactly one of two possible keys, labeled B_T and B_F. **(E) Step 4:** Bob picks up exactly one key B_b where *b* corresponds to his hidden input which only he knows (the oblivious transfer protocol ensures that Bob can only pick up *one* key). After Bob makes his selection, Alice shuffles the doubly-locked boxes and hands them to Bob along with the key A_a corresponding to her input a. Steps 1-4 is repeated for each of the inputs to the function. **(F) Step 5:** For each operator in the function that depends only on input values (i.e., the first "layer" of the circuit), Bob has four doubly-locked boxes and two keys A_a and B_b but he does not know Alice's input and Alice does not know Bob's input. He uses A_a and B_b and tries to unlock all four boxes. Only one of the four doubly-locked boxes will successfully open, revealing the joint output without revealing Alice's or Bob's inputs. **(G) Step 6:** The revealed output yields the key for the next operation (gate) in the circuit. Steps 5 and 6 are repeated for each operation in the function. At the end of the computation, instead of keys, Bob obtains the values that make up the output of the computation. **(H) Step 7:** The secure two-party computation process can be expanded to *N* parties by using additive secret sharing between two non-colluding cloud servers. Any (general) *N*-input function can be transformed into a 2-input function.

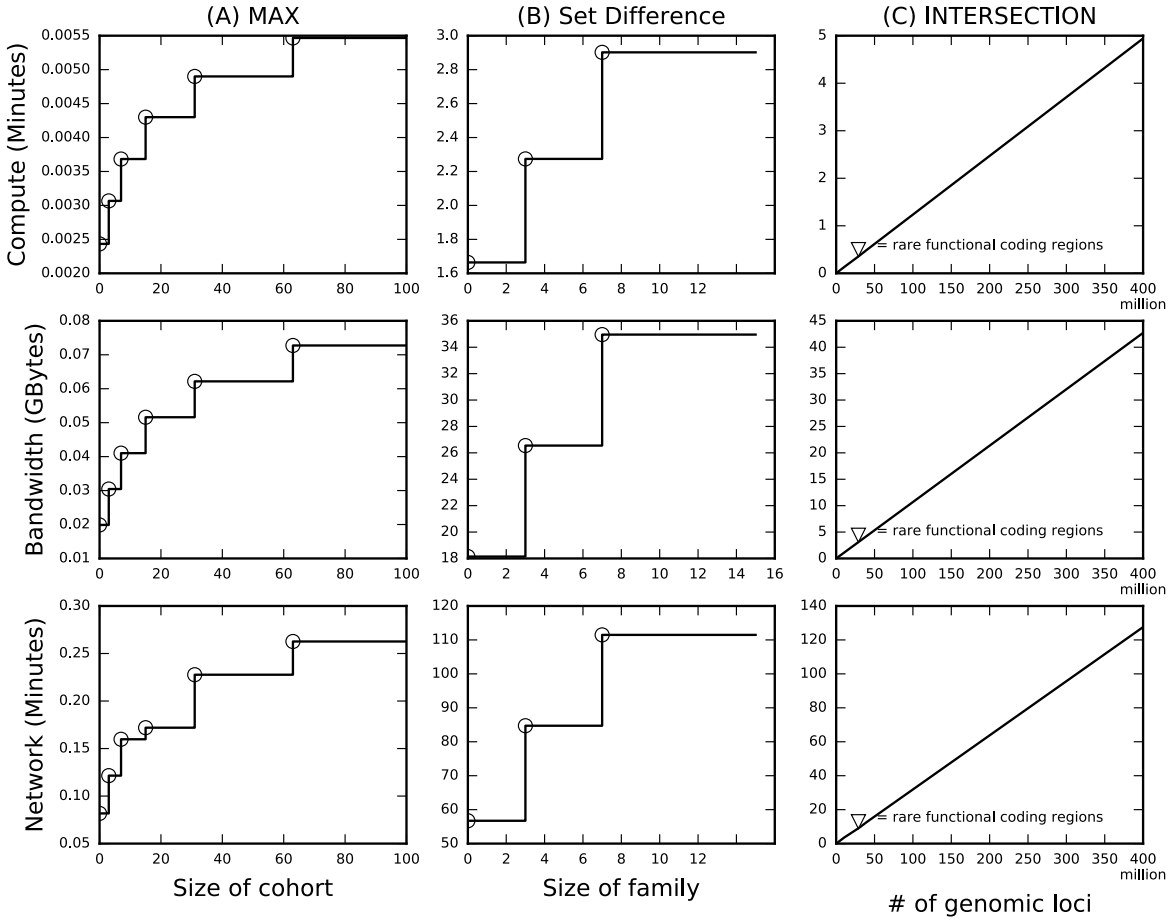
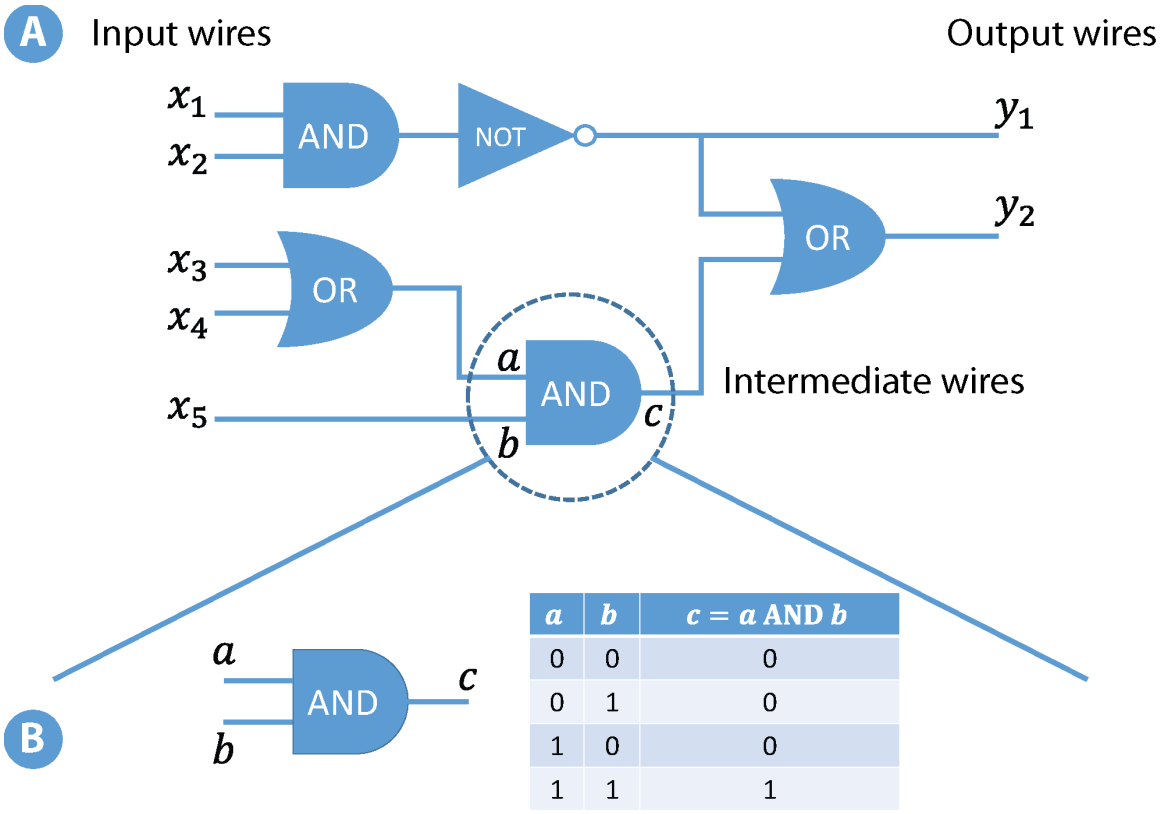


Fig. S3

Performance scale up for secure computation. Bandwidth, compute (CPU) time, and overall protocol execution (wall clock) time for the secure MAX, SETDIFF and INTERSECTION scenarios of Table 1, using a single thread on two servers, one located on the East Coast and the other on the West Coast. **(A)** When increasing the number of unrelated subjects in a small cohort study, all parameters grow logarithmically. **(B)** When increasing the number of family members in an affected / non-affected scenario, parameters also grow logarithmically. **(C)** In the two-hospital scenario, when increasing the number of genomic positions of potential interest (e.g., from the exome to the non-coding genome), all parameters grow linearly. Note that all three scenarios (A-C) perform the bulk of their computation on each element of the input vector independently (Fig. S1C). All scenarios can thus be parallelized for maximum speed-up using multiple threads and nodes.



a	b	$c = a$ AND b
0	0	0
0	1	0
1	0	0
1	1	1

Fig. S4

Computation using circuits. **(A)** A Boolean circuit consists of a sequence of logic gates (e.g., AND gates, OR gates, and NOT gates). Each logic gate takes one or two bits as input and produces a single bit of output. In a Boolean circuit, the outputs of one logic gate can be used as the input to another logic gate. We refer to these values as the *intermediate* values in the computation. In the circuit depicted in the figure, the inputs to the circuit are denoted x_1, x_2, x_3, x_4, x_5 and the outputs of the circuit are denoted y_1, y_2. Specifically, this particular circuit implements a function over five input bits and produces two output bits. **(B)** Each gate in the Boolean circuit can be described by a truth table that specifies the mapping between each configuration of the input bits to a corresponding output bit. In the case of an AND gate, there are two input bits, and the output is 1 if and only if both input bits are 1. Otherwise, the output is 0.

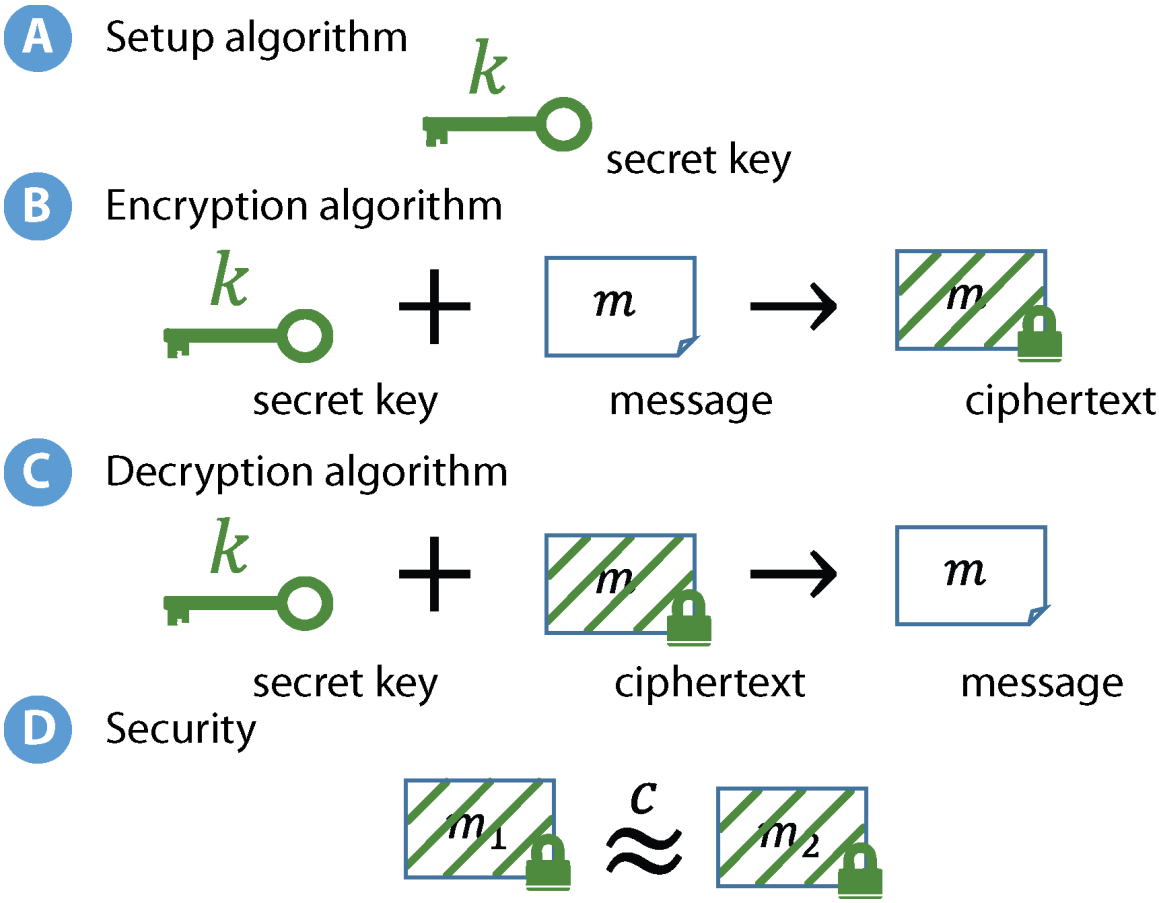


Fig. S5

Encryption and decryption overview. A secret-key encryption scheme consists of three algorithms: **(A)** a setup (or key-generation) algorithm which outputs a secret key (usually a long random string); **(B)** an encryption algorithm that takes in a secret key k and a message m and produces an encryption of m (called a ciphertext); and **(C)** a decryption algorithm that takes in the same secret key k and a ciphertext and produces the original message. We write Encrypt(k, m) to denote an encryption of the message m under the secret key k. The correctness requirement for an encryption scheme states that decrypting the ciphertext output by Encrypt(k, m) using the secret key k should yield the original message (plaintext) m. **(D)** The security requirement for a secret-key encryption scheme states that anyone who does *not* possess the secret-key k cannot distinguish an encryption of a message m_1 from an encryption of a message m_2, *irrespective* of the choice of messages m_1 and m_2. In other words, without the secret key, the ciphertext does not reveal any information about the encrypted message.

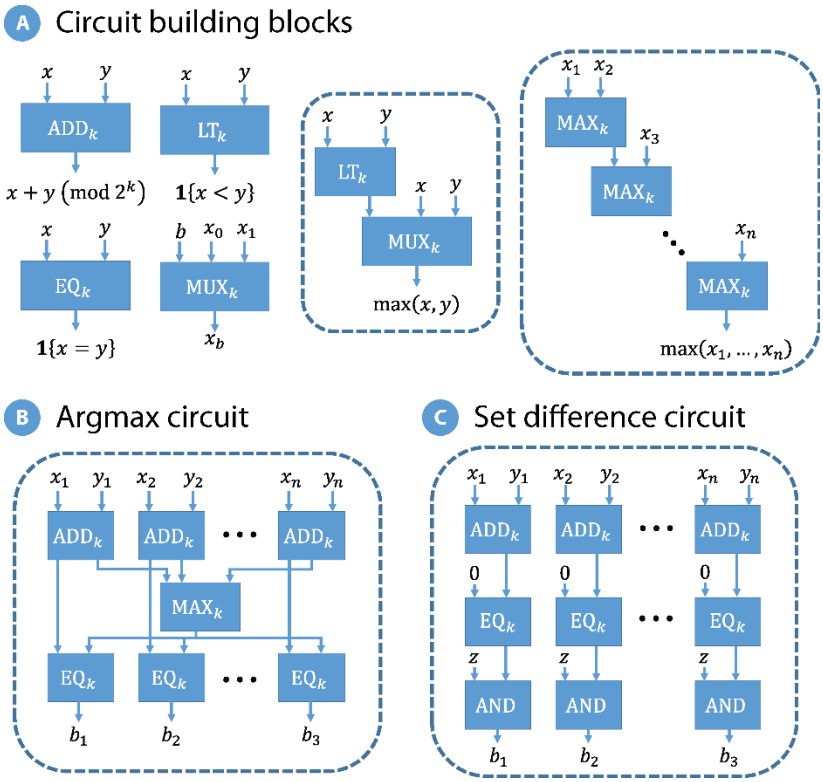


Fig. S6

Boolean building blocks for composing complex functions. **(A)** The basic building blocks we use to build our circuits for identifying common mutations and shared de novo variants include addition, comparison, equality, and multiplexer circuits. An addition circuit ADD_k on k bit inputs takes two k-bit values and outputs the k-bit representation of their sum (addition is performed modulo 2^k). The LT_k and EQ_k circuits implement the less-than and equality operations, respectively, on k-bit inputs. The MUX_k circuit implements a multiplexer circuit which on inputs a selection bit $b \in \{0,1\}$ and two k-bit values x_0, x_1, outputs x_b. The individual circuits can be efficiently constructed using AND gates and XOR gates, as described by Kolesnikov, Sadeghi, and Schneider (*47*). These basic circuit building blocks can be composed to build a max circuit MAX_k on two inputs (each of length k), which in turn can be used to build a max circuit on n inputs. **(B)** This circuit computes the argmax over n additively secret-shared values $z_1, \ldots, z_n$. The circuit operates by first combining the shares and then taking the max over the resulting vector of values. The argmax is represented by a bit-string of length n, where a position i has value 1 if z_i is equal to the max value, and 0 otherwise. **(C)** This circuit computes the set of genes (represented by indices) that are present in a test vector $z_1, \ldots, z_n$ but not present in a pool $t_1, \ldots, t_n$. The counts of the mutations appearing in the pool are additively secret shared. The circuit first combines the shares, and then identifies the indices i that appear in the test vector ($z_i = 1$), but not present in the pool ($t_i = 0$). The circuit outputs $b_i = 1$ if the gene indexed by i occurs in the target genome but not in the test pool, and 0 otherwise.

References and Notes

1. A. M. Wenger, H. Guturu, J. A. Bernstein, G. Bejerano, Systematic reanalysis of clinical exome data yields additional diagnoses: Implications for providers. *Genet. Med.* **19**, 209–214 (2017). doi:10.1038/gim.2016.88 Medline

2. A. M. Gazzo, D. Daneels, E. Cilia, M. Bonduelle, M. Abramowicz, S. Van Dooren, G. Smits, T. Lenaerts, DIDA: A curated and annotated digenic diseases database. *Nucleic Acids Res.* **44**, D900–D907 (2016). doi:10.1093/nar/gkv1068 Medline

3. I. M. Nolte, P. J. van der Most, B. Z. Alizadeh, P. I. W. de Bakker, H. M. Boezen, M. Bruinenberg, L. Franke, P. van der Harst, G. Navis, D. S. Postma, M. G. Rots, R. P. Stolk, M. A. Swertz, B. H. R. Wolffenbuttel, C. Wijmenga, H. Snieder, Missing heritability: Is the gap closing? An analysis of 32 complex traits in the Lifelines Cohort Study. *Eur. J. Hum. Genet.* **25**, 877–885 (2017). doi:10.1038/ejhg.2017.50 Medline

4. H. L. Rehm, S. J. Bale, P. Bayrak-Toydemir, J. S. Berg, K. K. Brown, J. L. Deignan, M. J. Friez, B. H. Funke, M. R. Hegde, E. Lyon; Working Group of the American College of Medical Genetics and Genomics Laboratory Quality Assurance Commitee, ACMG clinical laboratory standards for next-generation sequencing. *Genet. Med.* **15**, 733–747 (2013). doi:10.1038/gim.2013.92 Medline

5. M. Lek, K. J. Karczewski, E. V. Minikel, K. E. Samocha, E. Banks, T. Fennell, A. H. O'Donnell-Luria, J. S. Ware, A. J. Hill, B. B. Cummings, T. Tukiainen, D. P. Birnbaum, J. A. Kosmicki, L. E. Duncan, K. Estrada, F. Zhao, J. Zou, E. Pierce-Hoffman, J. Berghout, D. N. Cooper, N. Deflaux, M. DePristo, R. Do, J. Flannick, M. Fromer, L. Gauthier, J. Goldstein, N. Gupta, D. Howrigan, A. Kiezun, M. I. Kurki, A. L. Moonshine, P. Natarajan, L. Orozco, G. M. Peloso, R. Poplin, M. A. Rivas, V. Ruano-Rubio, S. A. Rose, D. M. Ruderfer, K. Shakir, P. D. Stenson, C. Stevens, B. P. Thomas, G. Tiao, M. T. Tusie-Luna, B. Weisburd, H.-H. Won, D. Yu, D. M. Altshuler, D. Ardissino, M. Boehnke, J. Danesh, S. Donnelly, R. Elosua, J. C. Florez, S. B. Gabriel, G. Getz, S. J. Glatt, C. M. Hultman, S. Kathiresan, M. Laakso, S. McCarroll, M. I. McCarthy, D. McGovern, R. McPherson, B. M. Neale, A. Palotie, S. M. Purcell, D. Saleheen, J. M. Scharf, P. Sklar, P. F. Sullivan, J. Tuomilehto, M. T. Tsuang, H. C. Watkins, J. G. Wilson, M. J. Daly, D. G. MacArthur; Exome Aggregation Consortium, Analysis of protein-coding genetic variation in 60,706 humans. *Nature* **536**, 285–291 (2016). doi:10.1038/nature19057 Medline

6. S. B. Ng, E. H. Turner, P. D. Robertson, S. D. Flygare, A. W. Bigham, C. Lee, T. Shaffer, M. Wong, A. Bhattacharjee, E. E. Eichler, M. Bamshad, D. A. Nickerson, J. Shendure, Targeted capture and massively parallel sequencing of 12 human exomes. *Nature* **461**, 272–276 (2009). doi:10.1038/nature08250 Medline

7. S. B. Ng, K. J. Buckingham, C. Lee, A. W. Bigham, H. K. Tabor, K. M. Dent, C. D. Huff, P. T. Shannon, E. W. Jabs, D. A. Nickerson, J. Shendure, M. J. Bamshad, Exome sequencing identifies the cause of a mendelian disorder. *Nat. Genet.* **42**, 30–35 (2010). doi:10.1038/ng.499 Medline

8. S. B. Ng, A. W. Bigham, K. J. Buckingham, M. C. Hannibal, M. J. McMillin, H. I. Gildersleeve, A. E. Beck, H. K. Tabor, G. M. Cooper, H. C. Mefford, C. Lee, E. H. Turner, J. D. Smith, M. J. Rieder, K. Yoshiura, N. Matsumoto, T. Ohta, N. Niikawa, D.

A. Nickerson, M. J. Bamshad, J. Shendure, Exome sequencing identifies MLL2 mutations as a cause of Kabuki syndrome. *Nat. Genet.* **42**, 790–793 (2010). doi:10.1038/ng.646 Medline

9. J. Kaiser, Baby genome screening needs more time to gestate. *Science* **354**, 398–399 (2016). doi:10.1126/science.354.6311.398 Medline

10. M. D. Mailman, M. Feolo, Y. Jin, M. Kimura, K. Tryka, R. Bagoutdinov, L. Hao, A. Kiang, J. Paschall, L. Phan, N. Popova, S. Pretel, L. Ziyabari, M. Lee, Y. Shao, Z. Y. Wang, K. Sirotkin, M. Ward, M. Kholodov, K. Zbicz, J. Beck, M. Kimelman, S. Shevelev, D. Preuss, E. Yaschenko, A. Graeff, J. Ostell, S. T. Sherry, The NCBI dbGaP database of genotypes and phenotypes. *Nat. Genet.* **39**, 1181–1186 (2007). doi:10.1038/ng1007-1181 Medline

11. L. L. Siu, M. Lawler, D. Haussler, B. M. Knoppers, J. Lewin, D. J. Vis, R. G. Liao, F. Andre, I. Banks, J. C. Barrett, C. Caldas, A. A. Camargo, R. C. Fitzgerald, M. Mao, J. E. Mattison, W. Pao, W. R. Sellers, P. Sullivan, B. T. Teh, R. L. Ward, J. C. ZenKlusen, C. L. Sawyers, E. E. Voest, Facilitating a culture of responsible and effective sharing of cancer genome data. *Nat. Med.* **22**, 464–471 (2016). doi:10.1038/nm.4089 Medline

12. S. S. Shringarpure, C. D. Bustamante, Privacy risks from genomic data-sharing beacons. *Am. J. Hum. Genet.* **97**, 631–646 (2015). doi:10.1016/j.ajhg.2015.09.010 Medline

13. A. Regalado, Internet of DNA: A global network of millions of genomes could be medicine's next great advance. *MIT Technol. Rev.* (2015); www.technologyreview.com/s/535016/internet-of-dna/.

14. M. A. Simpson, M. D. Irving, E. Asilmaz, M. J. Gray, D. Dafou, F. V. Elmslie, S. Mansour, S. E. Holder, C. E. Brain, B. K. Burton, K. H. Kim, R. M. Pauli, S. Aftimos, H. Stewart, C. A. Kim, M. Holder-Espinasse, S. P. Robertson, W. M. Drake, R. C. Trembath, Mutations in NOTCH2 cause Hajdu-Cheney syndrome, a disorder of severe and progressive bone loss. *Nat. Genet.* **43**, 303–305 (2011). doi:10.1038/ng.779 Medline

15. J.-B. Rivière, B. W. M. van Bon, A. Hoischen, S. S. Kholmanskikh, B. J. O'Roak, C. Gilissen, S. Gijsen, C. T. Sullivan, S. L. Christian, O. A. Abdul-Rahman, J. F. Atkin, N. Chassaing, V. Drouin-Garraud, A. E. Fry, J.-P. Fryns, K. W. Gripp, M. Kempers, T. Kleefstra, G. M. S. Mancini, M. J. M. Nowaczyk, C. M. A. van Ravenswaaij-Arts, T. Roscioli, M. Marble, J. A. Rosenfeld, V. M. Siu, B. B. A. de Vries, J. Shendure, A. Verloes, J. A. Veltman, H. G. Brunner, M. E. Ross, D. T. Pilz, W. B. Dobyns, De novo mutations in the actin genes *ACTB* and *ACTG1* cause Baraitser-Winter syndrome. *Nat. Genet.* **44**, 440–444 (2012). doi:10.1038/ng.1091 Medline

16. K. L. McBride, M. F. Riley, G. A. Zender, S. M. Fitzgerald-Butt, J. A. Towbin, J. W. Belmont, S. E. Cole, NOTCH1 mutations in individuals with left ventricular outflow tract malformations reduce ligand-induced signaling. *Hum. Mol. Genet.* **17**, 2886–2893 (2008). doi:10.1093/hmg/ddn187 Medline

17. S. Fenske, R. Mader, A. Scharr, C. Paparizos, X. Cao-Ehlker, S. Michalakis, L. Shaltiel, M. Weidinger, J. Stieber, S. Feil, R. Feil, F. Hofmann, C. Wahl-Schott, M. Biel, HCN3 contributes to the ventricular action potential waveform in the murine heart. *Circ. Res.* **109**, 1015–1023 (2011). doi:10.1161/CIRCRESAHA.111.246173 Medline

18. Y.-F. Huang, B. Gulko, A. Siepel, Fast, scalable prediction of deleterious noncoding variants from functional and population genomic data. *Nat. Genet.* **49**, 618–624 (2017). doi:10.1038/ng.3810 Medline

19. N. Sobreira, F. Schiettecatte, D. Valle, A. Hamosh, GeneMatcher: A matching tool for connecting investigators with an interest in the same gene. *Hum. Mutat.* **36**, 928–930 (2015). doi:10.1002/humu.22844 Medline

20. A. A. Philippakis, D. R. Azzariti, S. Beltran, A. J. Brookes, C. A. Brownstein, M. Brudno, H. G. Brunner, O. J. Buske, K. Carey, C. Doll, S. Dumitriu, S. O. M. Dyke, J. T. den Dunnen, H. V. Firth, R. A. Gibbs, M. Girdea, M. Gonzalez, M. A. Haendel, A. Hamosh, I. A. Holm, L. Huang, M. E. Hurles, B. Hutton, J. B. Krier, A. Misyura, C. J. Mungall, J. Paschall, B. Paten, P. N. Robinson, F. Schiettecatte, N. L. Sobreira, G. J. Swaminathan, P. E. Taschner, S. F. Terry, N. L. Washington, S. Züchner, K. M. Boycott, H. L. Rehm, The Matchmaker Exchange: A platform for rare disease gene discovery. *Hum. Mutat.* **36**, 915–921 (2015). doi:10.1002/humu.22858 Medline

21. Y. Erlich, A. Narayanan, Routes for breaching and protecting genetic privacy. *Nat. Rev. Genet.* **15**, 409–421 (2014). doi:10.1038/nrg3723 Medline

22. M. S. Riazi, N. K. R. Dantu, L. N. V. Gattu, F. Koushanfar, in *Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* (IEEE, 2016), pp. 248–253.

23. Y. Lindell, B. Pinkas, A proof of security of Yao's protocol for two-party computation. *J. Cryptol.* **22**, 161–188 (2009). doi:10.1007/s00145-008-9036-8

24. C. Hazay, Y. Lindell, *Efficient Secure Two-Party Protocols - Techniques and Constructions* (Information Security and Cryptography Series, Springer, 2010).

25. C. Dwork, in *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming* (Springer, 2006), pp. 1–12.

26. S. Simmons, C. Sahinalp, B. Berger, Enabling privacy-preserving GWASs in heterogeneous human populations. *Cell Syst.* **3**, 54–61 (2016). doi:10.1016/j.cels.2016.04.013 Medline

27. F. Chen, S. Wang, X. Jiang, S. Ding, Y. Lu, J. Kim, S. C. Sahinalp, C. Shimizu, J. C. Burns, V. J. Wright, E. Png, M. L. Hibberd, D. D. Lloyd, H. Yang, A. Telenti, C. S. Bloss, D. Fox, K. Lauter, L. Ohno-Machado, PRINCESS: Privacy-protecting Rare disease International Network Collaboration via Encryption through Software guard extensionS. *Bioinformatics* **33**, 871–878 (2017). Medline

28. C. Gentry, in *Proceedings of the 41st ACM Symposium on Theory of Computing* (ACM, 2009), pp. 169–178.

29. N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, J. Wernsing, Manual for using homomorphic encryption for bioinformatics. *Proc. IEEE* **105**, 552–567 (2017).

30. I. Chillotti, N. Gama, M. Georgieva, M. Izabachène, in *Proceedings of the 22nd Annual International Conference on the Theory and Applications of Cryptology and Information* (Springer, 2016), pp. 3–33.

31. H. Li, R. Durbin, Fast and accurate long-read alignment with Burrows-Wheeler transform. *Bioinformatics* **26**, 589–595 (2010). doi:10.1093/bioinformatics/btp698 Medline

32. A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, M. A. DePristo, The Genome Analysis Toolkit: A MapReduce framework for analyzing next-generation DNA sequencing data. *Genome Res.* **20**, 1297–1303 (2010). doi:10.1101/gr.107524.110 Medline

33. K. Wang, M. Li, H. Hakonarson, ANNOVAR: Functional annotation of genetic variants from high-throughput sequencing data. *Nucleic Acids Res.* **38**, e164 (2010). doi:10.1093/nar/gkq603 Medline

34. F. Cunningham, M. R. Amode, D. Barrell, K. Beal, K. Billis, S. Brent, D. Carvalho-Silva, P. Clapham, G. Coates, S. Fitzgerald, L. Gil, C. G. Girón, L. Gordon, T. Hourlier, S. E. Hunt, S. H. Janacek, N. Johnson, T. Juettemann, A. K. Kähäri, S. Keenan, F. J. Martin, T. Maurel, W. McLaren, D. N. Murphy, R. Nag, B. Overduin, A. Parker, M. Patricio, E. Perry, M. Pignatelli, H. S. Riat, D. Sheppard, K. Taylor, A. Thormann, A. Vullo, S. P. Wilder, A. Zadissa, B. L. Aken, E. Birney, J. Harrow, R. Kinsella, M. Muffato, M. Ruffier, S. M. J. Searle, G. Spudich, S. J. Trevanion, A. Yates, D. R. Zerbino, P. Flicek, Ensembl 2015. *Nucleic Acids Res.* **43**, D662–D669 (2015). doi:10.1093/nar/gku1010 Medline

35. A. C.-C. Yao, in *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science* (IEEE, 1982), pp. 160–164.

36. O. Goldreich, S. Micali, A. Wigderson, in *Proceedings of the 19th ACM Symposium on Theory of Computing* (ACM, 1987), pp. 218–229.

37. J. Katz, Y. Lindell, *Introduction to Modern Cryptography* (Chapman and Hall/CRC Press, 2007).

38. M. O. Rabin, "How to exchange secrets with oblivious transfer" (Technical Report TR-81, Aiken Computation Lab, Harvard Univ., 1981); https://eprint.iacr.org/2005/187.

39. J. Kilian, in *Proceedings of the 20th ACM Symposium on Theory of Computing* (ACM, 1988), pp. 20–31.

40. M. Naor, B. Pinkas, in *Proceedings of the 31st ACM Symposium on Theory of Computing* (ACM, 1999), pp. 245–254.

41. Y. Lindell, B. Pinkas, Secure multiparty computation for privacy-preserving data mining. *J. Priv. Confidentiality* **1**, 59–98 (2009).

42. M. Bellare, V. T. Hoang, S. Keelveedhi, P. Rogaway, in *IEEE Symposium on Security and Privacy* (IEEE, 2013), pp. 478–492.

43. V. Kolesnikov, T. Schneider, in *Proceedings of the 35th International Colloquium on Automata, Languages and Programming* (Springer, 2008), pp. 486–498.

44. S. Zahur, M. Rosulek, D. Evans, in *Proceedings of the 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (Springer, 2015), pp. 220–250.

45. I. Damgard, V. Pastro, N. P. Smart, S. Zakarias, in *Proceedings of the 32nd International Cryptology Conference* (Springer, 2012), pp. 643–662.

46. M. Ben-Or, S. Goldwasser, A. Wigderson, in *Proceedings of the 20th ACM Symposium on Theory of Computing* (ACM, 1988), pp. 1–10.

47. V. Kolesnikov, A.-R. Sadeghi, T. Schneider, in *Proceedings of the 8th International Conference on Cryptology and Network Security* (Springer, 2009), pp. 1–20.

48. G. Asharov, Y. Lindell, T. Schneider, M. Zohner, in *Proceedings of the 2013 ACM Conference on Computer and Communications Security* (ACM, 2013), pp. 535–548.